

1

2 **P1788.1/D9.8**
3 **Draft Standard for Interval Arithmetic (Simplified)**
4

5 Sponsor

6 **Microprocessor Standards Committee**

7 of the

8 **IEEE Computer Society**

1 **Abstract:**

2 This standard is a simplified version and a subset of the IEEE Std 1788TM-2015 for Interval Arithmetic and
3 includes those operations and features of the latter that in the the editors' view are most commonly used in
4 practice. IEEE P1788.1 specifies interval arithmetic operations based on intervals whose endpoints are IEEE 754
5 binary64 floating-point numbers and a decoration system for exception-free computations and propagation of
6 properties of the computed results.

7 A program built on top of an implementation of IEEE P1788.1 should compile and run, and give identical output
8 within round off, using an implementation of IEEE Std 1788TM-2015, or any superset of the former.

9 Compared to IEEE Std 1788TM-2015, this standard aims to be minimalistic, yet to cover much of the functionality
10 needed for interval computations. As such, it is more accessible and will be much easier to implement, and thus
11 will speed up production of implementations.

12 **Keywords:** arithmetic, computing, decoration, enclosure, interval, operation, verified, IEEE Std 1788TM-2015

13 _____
14 The Institute of Electrical and Electronics Engineers, Inc.
15 3 Park Avenue, New York, NY 10016-5997, USA

16 Copyright ©2014 by The Institute of Electrical and Electronics Engineers, Inc.
17 All rights reserved. Published 0. Printed in the United States of America.

18 IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers,
19 Incorporated.

20 PDF: ISBN 978-0-XXXX-XXXX-X STDXXXXX
21 Print: ISBN 978-0-XXXX-XXXX-X STDPDXXXXX

22 *IEEE prohibits discrimination, harassment, and bullying.*

23 *For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>.*

24 *No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written
25 permission of the publisher.*

1 **Important Notices and Disclaimers Concerning IEEE Standards Documents**

2 IEEE documents are made available for use subject to important notices and legal disclaimers. These notices
3 and disclaimers, or a reference to this page, appear in all standards and may be found under the heading
4 “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

5 **Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents**

6 IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use,
7 are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards
8 Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consen-
9 sus development process, approved by the American National Standards Institute (“ANSI”), which brings
10 together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers
11 are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE
12 administers the process and establishes rules to promote fairness in the consensus development process, IEEE
13 does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of
14 any judgments contained in its standards.

15 IEEE does not warrant or represent the accuracy or content of the material contained in its standards,
16 and expressly disclaims all warranties (express, implied and statutory) not included in this or any other
17 document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness
18 for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of
19 material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort.
20 IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

21 Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there
22 are no other ways to produce, test, measure, purchase, market, or provide other goods and services related
23 to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved
24 and issued is subject to change brought about through developments in the state of the art and comments
25 received from users of the standard.

26 In publishing and making its standards available, IEEE is not suggesting or rendering professional or other
27 services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any
28 other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon
29 his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as
30 appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE
31 standard.

32 IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
33 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCURE-
34 MENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
35 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CON-
36 TRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
37 ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN
38 IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH
39 DAMAGE WAS FORESEEABLE.

40 **Translations**

41 The IEEE consensus development process involves the review of documents in English only. In the event
42 that an IEEE standard is translated, only the English version published by IEEE should be considered the
43 approved IEEE standard.

44 **Official statements**

45 A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Oper-
46 ations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees
47 and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia,
48 seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear

1 that his or her views should be considered the personal views of that individual rather than the formal
2 position of IEEE.

3 **Comments on standards**

4 Comments for revision of IEEE Standards documents are welcome from any interested party, regardless
5 of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice
6 pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a
7 proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a
8 consensus of concerned interests, it is important that any responses to comments and questions also receive
9 the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards
10 Coordinating Committees are not able to provide an instant response to comments or questions except in
11 those cases where the matter has previously been addressed. For the same reason, IEEE does not respond
12 to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is
13 welcome to join the relevant IEEE working group.

14 Comments on standards should be submitted to the following address:

15 Secretary, IEEE-SA Standards Board
16 445 Hoes Lane
17 Piscataway, NJ 08854 USA

18 **Laws and regulations**

19 Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with
20 the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory
21 requirements. Implementers of the standard are responsible for observing or referring to the applicable
22 regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is
23 not in compliance with applicable laws, and these documents may not be construed as doing so.

24 **Copyrights**

25 IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws.
26 They are made available by IEEE and are adopted for a wide variety of both public and private uses. These
27 include both use, by reference, in laws and regulations, and use in private self-regulation, standardization,
28 and the promotion of engineering practices and methods. By making these documents available for use
29 and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the
30 documents.

31 **Photocopies**

32 Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy
33 portions of any individual standard for company or organizational internal use or individual, non-commercial
34 use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer
35 Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of
36 any individual standard for educational classroom use can also be obtained through the Copyright Clearance
37 Center.

38 **Updating of IEEE Standards documents**

39 Users of IEEE Standards documents should be aware that these documents may be superseded at any time
40 by the issuance of new editions or may be amended from time to time through the issuance of amendments,
41 corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the
42 document together with any amendments, corrigenda, or errata then in effect.

43 Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years
44 old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of
45 some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that
46 they have the latest edition of any IEEE standard.

1 In order to determine whether a given document is the current edition and whether it has been amended
2 through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at [http://ieeexplore.
4 ieee.org/xpl/standards.jsp](http://ieeexplore.
3 ieee.org/xpl/standards.jsp) or contact IEEE at the address listed previously. For more information about
5 the IEEE SA or IEEE's standards development process, visit the IEEE-SA Website at [http://standards.
7 ieee.org](http://standards.
6 ieee.org).

6 **Errata**

7 Errata, if any, for all IEEE standards can be accessed on the IEEE-SA Website at the following URL:
8 <http://standards.ieee.org/findstds/errata/index.html>. Users are encouraged to check this URL for
9 errata periodically.

10 **Patents**

11 Attention is called to the possibility that implementation of this standard may require use of subject matter
12 covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect
13 to the existence or validity of any patent rights in connection therewith. If a patent holder or patent
14 applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed
15 on the IEEE-SA Website at <http://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of
16 Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights
17 without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably
18 free of any unfair discrimination to applicants desiring to obtain such licenses.

19 Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not re-
20 sponsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries
21 into the legal validity or scope of Patent Claims, or determining whether any licensing terms or conditions
22 provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are
23 reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the
24 validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility.
25 Further information may be obtained from the IEEE Standards Association.

1 **Participants**

2 At the time this draft IEEE standard was completed, the Interval Standard Working Group had the following
3 membership:

4 **Nathalie Revol**, *Chair*
5 **R. Baker Kearfott**, *Vice Chair*
6 **Guillaume Melquiond**, *Secretary*
7 **J. Wolff Von Gudenberg**, *Web Master*
8 **George Corliss**, *Vote Tabulator*
9 **Ned Nedialkov**, *Technical Editor*
10 **John Pryce**, *Technical Editor*
11 **Christian Keil**, *Deputy Technical Editor*
12 **Michel Hack**, **Vincent Lefevre**, **Ian McIntosh**, **Dmitry Nadezhin**,
13 **and J. Wolff Von Gudenberg**, *Assistant Technical Editors*

Alexandru Amaricai	Ralph Kearfott	Antony Popov
Ayman Bakr	Christian Keil	Evgenija Popova
Ahmed Belhani	Michel Kieffer	John Pryce
Gerd Bohlender	Walter Kraemer	Tarek Raissi
Gilles Chabert	Vladik Kreinovich	Nacim Ramdani
George Corliss	Ulrich Kulisch	Andreas Rauh
Rudnei Dias da Cunha	Dorina Lanza	Gaby Dos Reis
Hend Dawood	Vincent Lefevre	Nathalie Revol
William Edmonson	David Lester	Michael Schulte
Bo Einarsson	Dominique Lohez	Kyarash Shahriari
Alan Eliassen	Wolfram Luther	Stefan Siegel
Hossam Fahmy	Amin Maher	Iwona Skalna
Richard Fateman	Svetoslav Markov	Mark Stadtherr
Scott Ferson	Guenter Mayer	James Stine
Haitham Gad	R. Ian McIntosh	Pipop Thienprapasith
14 Kathy Gerber	Guillaume Melquiond	Warwick Tucker
Alexandre Goldsztejn	Jean-Pierre Merlet	Alfredo Vaccaro
Frederic Goualard	Jean-Michel Muller	Maarten van Emden
Michael Groszkiewicz	Humberto Munoz	Erik-Jan van Kampen
Mohamed Guerfel	Jose Antonio Munoz	Van Snyder
Michel Hack	Dima Nadezhin	Josep Vehi
Robert Hanek	Kaori Nagatou	Julio Villalba-Moreno
Behnam Hashemi	Mitsuhiro Nakao	J. Wolff Von Gudenberg
Nathan Hayes	Ned Nedialkov	G. William Walster
Oliver Heimlich	Markus Neher	Yan Wang
Timothy Hickey	Marco Nehmeier	Lee Winter
Werner Hofschuster	Diep Nguyen	Pierre-Alain Yvars
Chenyi Hu	Michael Nooner	Sergei Zhilin
Trevor Jackson-III	Shinichi Oishi	Mohamed Zidan
Malgorzata Jankowska	Sylvain Pion	Dan Zuras

1 Introduction

2 This introduction is not part of P1788/D9.8, Draft Standard for Interval Arithmetic (Simplified).

3 This standard is a simplified version of the (full) IEEE Std 1788TM-2015 for Interval Arithmetic. As such,
4 IEEE P1788.1 features

- 5 a) set-based intervals,
- 6 b) the inf-sup interval type based on the IEEE 754 binary64 floating-point format,
- 7 c) the decoration system, and
- 8 d) simplified I/O.

9 The full standard extends this standard with extra features, aimed at current applications of interval com-
10 putation and at anticipated future developments, such as the following.

- 11 – Finite-precision intervals of different kinds are supported, e.g. one can build a system where interval
12 endpoints are numbers from an arbitrary-precision floating-point system such as MPFR.
- 13 – Besides the model in 1788.1, where a mathematical interval is any closed connected subset of the real
14 numbers, other foundational models are supported in a controlled way by the concept of *flavors*. For
15 instance a flavor may support half-open and open real intervals; or Kaucher/modal intervals like [4,3] as
16 well as [3,4].
- 17 IEEE P1788.1 satisfies all General Requirements of IEEE Std 1788TM-2015 and can be considered a flavor
18 of that standard.
- 19 – The set of required operations is extended by operations useful in specialized applications, such as reverse-
20 mode functions for constraint programming, and two-output division used in the interval Newton root-
21 finding method.

22 This document consists of two parts. Part 1 contains general requirements: overview, scope and purpose of
23 this standard and various definitions and abbreviations. Part 2 is the actual standard, presented in four levels.
24 Level 1 summarizes the theory of set-based intervals including decorations; Level 2 is about representing Level
25 1 entities in finite precision and the corresponding operations; Level 3 represents Level 2 entities, and Level
26 4 specifies interchange encodings. Annex A lists the features of IEEE Std 1788TM-2015 that are not required
27 in IEEE P1788.1.

1	Contents	
2	Part 1. General Requirements	2
3	1. Overview	2
4	1.1. Scope	2
5	1.2. Purpose	2
6	1.3. Word usage	2
7	1.4. Structure of the standard in levels	3
8	1.5. The meaning of conformance	3
9	2. Normative references	3
10	3. Special terms, abbreviations, and notation	3
11	3.1. Frequently used notation and abbreviations	3
12	3.2. Special terms	3
13	Part 2. Interval Standard (Simplified)	7
14	4. Level 1 description	7
15	4.1. Level 1 entities	7
16	4.2. Intervals	7
17	4.3. Hull	8
18	4.4. Functions	8
19	4.5. Required operations	9
20	5. The decoration system at Level 1	14
21	5.1. Overview	14
22	5.2. Definitions and properties	14
23	5.3. The ill-formed interval	15
24	5.4. Permitted combinations	15
25	5.5. Operations on/with decorations	15
26	5.6. Decorations and arithmetic operations	16
27	5.7. Decoration of non-arithmetic operations	16
28	6. Level 2 description	18
29	6.1. Introduction	18
30	6.2. Naming conventions for operations	18
31	6.3. Level 2 hull operation	19
32	6.4. Level 2 interval extensions	19
33	6.5. Accuracy of operations	19
34	6.6. Number and interval literals	20
35	6.7. Required operations	21
36	6.8. Input and output (I/O) of intervals	25
37	7. Levels 3 and 4 description	26
38	7.1. Representation	26
39	7.2. Operations and representation	26
40	7.3. Interchange representations and encodings	26
41	Annex A. Features of the full standard IEEE Std 1788 TM -2015 not required in the IEEE P1788.1	
42	standard (informative)	28

1 **P1788.1™/D9.8**

2 **Draft Standard for Interval Arithmetic (Simplified)**

3

4 *IMPORTANT NOTICE: IEEE Standards documents are not intended to ensure safety, secu-*
5 *rity, health, or environmental protection, or ensure against interference with or from other*
6 *devices or networks. Implementers of IEEE Standards documents are responsible for determin-*
7 *ing and complying with all appropriate safety, security, environmental, health, and interference*
8 *protection practices and all applicable laws and regulations.*

9 *This IEEE document is made available for use subject to important notices and legal dis-*
10 *claimers. These notices and disclaimers appear in all publications containing this document*
11 *and may be found under the heading Important Notice or Important Notices and Disclaimers*
12 *Concerning IEEE Documents. They can also be obtained on request from IEEE or viewed at*
13 *<http://standards.ieee.org/IPR/disclaimers.html>.*

General Requirements

1

1. Overview

1.1 Scope

This standard is a simplified version and a subset of the IEEE Std 1788TM-2015 for Interval Arithmetic and includes those operations and features of the latter that in the the editors' view are most commonly used in practice. IEEE P1788.1 specifies interval arithmetic operations based on intervals whose endpoints are IEEE 754 binary64 floating-point numbers and a decoration system for exception-free computations and propagation of properties of the computed results.

A program built on top of an implementation of IEEE P1788.1 should compile and run, and give identical output within round off, using an implementation of IEEE Std 1788TM-2015, or any superset of the former.

1.2 Purpose

Compared to IEEE Std 1788TM-2015, this standard aims to be minimalistic, yet to cover much of the functionality needed for interval computations. As such, it is more accessible and will be much easier to implement, and thus will speed up production of implementations.

1.3 Word usage

In this document 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 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:

- **optional** indicates features that may be omitted, but if provided shall be provided exactly as specified;
- **can** is used for statements of possibility and capability (“can” means “is able to”);
- **might** indicates the possibility of a situation that could occur, with no implication of the likelihood of that situation (“might” means “could possibly”);
- **comprise** indicates members of a set or list are exactly those objects having some property. An unqualified **consist of** merely asserts all members of a set have some property, e.g., “a binary floating-point format consists of numbers with a terminating binary representation”. “Comprises” means “consists exactly of”.
- **Example** introduces text that is informative (is not a requirement of this standard). Such text is set in sanserif font within brackets. [Example. Like this.]

1 1.4 Structure of the standard in levels

2 This standard is structured into four levels.

3 Level 1, *mathematics* (Clause 4), defines the underlying theory. The entities at this level are set-based
4 intervals and operations on them. This level defines a *decorated* interval, comprising a (bare) interval and a
5 *decoration* (Clause 5).

6 Level 2, *discretization* (Clause 6), approximates the mathematical theory by intervals with upper and lower
7 bounds that are IEEE 754 binary64 numbers and operations on such intervals. A level 2 entity is called a
8 *datum*.¹

9 Level 3, *representation* (Clause 7), represents interval datums in terms of binary64 values. A level 3 entity is
10 called an *interval object*. Representation of decorated intervals is also defined at this level.

11 Level 4, *encoding* (Clause 7), specifies encoding of interval objects into bit strings.

12 1.5 The meaning of conformance

13 An implementation of this standard shall satisfy the following requirements:

14 – provide the decorations specified in 5.2;

15 – provide implementations of the required operations in 6.7; required and recommended accuracies for these
16 operations are in 6.5;

17 – provide input and output functions to convert intervals from and to strings as specified in 6.8.2 and 6.8.3;
18 and

19 – provide an interchange representation as specified in 7.3.

20 2. Normative references

21 The following referenced documents are indispensable for the application of this document (i.e., they must
22 be understood and used, so each referenced document is cited in text and its relationship to this document
23 is explained).

24 IEEE Std 754TM-2008, IEEE Standard for Floating-Point Arithmetic²³

25 IEEE Std 1788TM-2015, IEEE Standard for Interval Arithmetic

26 3. Special terms, abbreviations, and notation

27 3.1 Frequently used notation and abbreviations

28 Frequently used notation and abbreviations are given in Table 3.1.

29 3.2 Special terms

30 **accuracy mode** A way to describe the quality of an interval version of a function. See also **tightness**.

31 NOTE—Details in 6.5.

32 **arithmetic operation** A version of a **point operation**.

33 NOTE—Details in 4.4.1.

34 **bare interval** Same as **interval**; used to emphasize it is not decorated.

35 NOTE—Details in 5.2.

¹Plural “datums” in this standard, since “data” is often misleading.

²IEEE publications are available from The Institute of Electrical and Electronics Engineers at <http://standards.ieee.org>.

³The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

Table 3.1. Frequently used notation and abbreviations

IA	Interval Arithmetic
\mathbb{R}	the set of real numbers
$\overline{\mathbb{R}}$	the set of extended real numbers, $\mathbb{R} \cup \{-\infty, +\infty\}$
\mathbb{IR}	the set of closed real intervals, including unbounded intervals and the empty set
\mathbb{T}	generic notation for an interval type
\emptyset , Empty	the empty set
Entire	the whole real line
NaN	Not an Interval
NaN	Not a Number
x, y, \dots [resp. f, g, \dots]	typeface/notation for a numeric value [resp. numeric function]
$\mathbf{x}, \mathbf{y}, \dots$ [resp. $\mathbf{f}, \mathbf{g}, \dots$]	typeface/notation for an interval value [resp. interval function]
f, g, \dots	typeface/notation for an expression, producing a function by evaluation
$\text{Dom}(f)$	the domain of a point-function f
$\text{Rge}(f \mathbf{s})$	the range of a point-function f over a set \mathbf{s} ; the same as the image of \mathbf{s} under f

1 **bounds** If \mathbf{x} is an interval then $\underline{x} = \inf \mathbf{x}$ and $\bar{x} = \sup \mathbf{x}$, its **lower bound** and **upper bound** respectively,
 2 are extended-real numbers that for a nonempty interval satisfy $\underline{x} \leq \bar{x}$, $\underline{x} < +\infty$ and $\bar{x} > -\infty$. By convention
 3 the empty set has lower bound $+\infty$ and upper bound $-\infty$. Note $\pm\infty$ can be bounds of an interval but never
 4 members of it.

5 **box** A **box** or **interval vector** is an n -dimensional interval, i.e. a tuple $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ where the \mathbf{x}_i are
 6 intervals. When an interval may be regarded as a set of points, it may be identified with the cartesian
 7 product $\mathbf{x}_1 \times \dots \times \mathbf{x}_n$.

8 **constant** A **real constant** at Level 1 is defined to be a scalar point function with no arguments; this
 9 includes the constant NaN.

10 **datum** One of the entities manipulated by finite precision (Level 2) operations. It can be a **boolean**, a
 11 **decoration**, a (bare) **interval**, a **decorated interval**, a **number** or a **string** datum.

12 NOTE—Details in 6.1.

13 **decoration** Decorations are used to record events in interval operations: e.g., evaluating a function at points
 14 where it is undefined. One of the five values **com**, **dac**, **def**, **trv** and **ill**.

15 NOTE—Details in Clause 5.

16 **decorated interval** A pair (interval, decoration).

17 NOTE—Details in Clause 5.

18 **domain** For a function with arguments taken from some set, the **domain** comprises those points in the set
 19 at which the function has a value. The domain of a point operation is part of its definition. E.g., the (point)
 20 operation of division x/y , in this standard, has arguments (x, y) in \mathbb{R}^2 , and its domain is $\{(x, y) \in \mathbb{R}^2 \mid$
 21 $y \neq 0\}$.

22 **elementary function** Synonymous with **arithmetic operation**.

23 **exception** An **exception** is an event that occurs, and may be signaled, when an operation on some partic-
 24 ular operands has no outcome suitable for every reasonable application. Exceptions may occur in interval
 25 constructors and in the **intervalPart** operation.

26 NOTE—Details in 6.1.4, 6.7.8.

27 **expression** A symbolic form used to define a function. An **arithmetic expression** is one whose operations
 28 are all arithmetic operations.

29 NOTE—Details in Clause 6 of IEEE Std 1788TM-2015.

30 **fma** Fused multiply-add operation that computes $x \times y + z$.

31 **FTIA, FTDA** For a real function f defined by an arithmetic expression:

- 1 – The Fundamental Theorem of Interval Arithmetic (FTIA) gives relations between the result \mathbf{y} of evaluating
2 the expression in interval mode over an input box and the behavior of f on the box. The basic property
3 is that \mathbf{y} contains the range of f over the box; various extra conditions can give extra conclusions, such as
4 that f is continuous on the box.
- 5 – The Fundamental Theorem of Decorated Interval Arithmetic (FTDIA) describes how evaluating the ex-
6 pression in decorated interval mode enables the various conditions of the FTIA to be verified, making the
7 corresponding conclusions computable.
8
- 9 NOTE—Details in 6.4 of IEEE Std 1788TM-2015.
- 10 **function** Has the usual mathematical meaning of a (possibly partial, i.e., not everywhere defined) function.
11 Synonymous with **map**, **mapping**.
- 12 **hull** (or **interval hull**) The hull of a subset s of \mathbb{R} is the tightest interval containing s .
- 13 **implementation** When used without qualification, means a realization of an interval arithmetic conforming
14 to the specification of this standard.
- 15 **inf-sup** Describes a representation of an interval based on its lower and upper bounds.
- 16 **interval** At Level 1, a (bare) interval \mathbf{x} is a closed connected subset of \mathbb{R} . At Level 2 a \mathbb{T} -interval is a
17 member of the bare interval type \mathbb{T} , or a non-NaI member of the decorated interval type \mathbb{T} .
18 NOTE—Details in 4.2, 6.
- 19 **interval extension** At Level 1, an interval extension of a point function f is a function \mathbf{f} from intervals to
20 intervals such that $f(x)$ belongs to $\mathbf{f}(\mathbf{x})$ whenever x belongs to \mathbf{x} and $f(x)$ is defined. It is the **natural** (or
21 tightest) interval extension, if $\mathbf{f}(\mathbf{x})$ is the interval hull of the range of f over \mathbf{x} , for all \mathbf{x} .
22 NOTE—Details in 4.4.4, for Level 2 interval extension, see 6.4.
- 23 **decorated interval extension** of f is a function from decorated intervals to decorated intervals, whose
24 interval part is an interval extension of f , and whose decoration part propagates decorations as specified in
25 5.6.
26 NOTE—Details in Clause 5.
- 27 **interval vector** See **box**.
- 28 **library** The set of Level 1 operations (Level 1 library) or those provided by an implementation (Level 2
29 library). Further classification may be made into the **point library**, **bare interval library** and **decorated**
30 **interval library**.
- 31 **map, mapping** See **function**.
- 32 **mathematical interval of constructor** The arguments of an interval constructor, if valid, define a math-
33 ematical interval \mathbf{x} . The actual interval returned by the constructor is the tightest interval that contains \mathbf{x} .
34 NOTE—Details in 6.7.5.
- 35 **NaI, NaN** At Level 1 NaN is the function $\mathbb{R}^0 \rightarrow \mathbb{R}$ with empty domain. NaI is the natural interval extension
36 of NaN, the map from subsets of \mathbb{R}^0 to subsets of \mathbb{R} whose value is always the empty set. At Level 2 NaN is
37 the Not a Number datum. NaI is the Not an Interval datum.
38 NOTE—Details in 5.3.
- 39 **no value** A mathematical (Level 1) operation evaluated at a point outside its domain is said to have no
40 value. Used instead of “undefined”, which can be ambiguous. E.g., in this standard, real number division
41 x/y has no value when $y = 0$.
42 NOTE—Details in 6.1.4.
- 43 **non-arithmetic operation** An operation on intervals that is not an interval version of a point function;
44 includes intersection and convex hull of two intervals.
45 NOTE—Details in 5.7.
- 46 **number** Any member of the set $\mathbb{R} \cup \{-\infty, +\infty\}$ of extended reals: a **finite number** if it belongs to \mathbb{R} , else
47 an **infinite number**.

- 1 **octet** Bit string of length 8, equivalently 8-bit byte.
- 2 An **octet-encoding** is a mapping from the conceptual bit string encodings of floating-point datums and of
3 decorations into an octet sequence.
- 4 NOTE—Details in 7.3.
- 5 **point function** A mathematical (Level 1) function of real variables.
- 6 NOTE—Details in 4.4.1.
- 7 **range** The range, $\text{Rge}(f | \mathbf{s})$, of a function f over a set \mathbf{s} is the set of values $f(x)$ at those points of \mathbf{s} where
8 f is defined.
- 9 NOTE—Details in 4.4.2.
- 10 **string, text** A **text string**, or just **string**, is a finite character sequence belonging to some alphabet. The
11 term **text** is also used to mean strings generally.
- 12 **tightest** Smallest in the partial order of set containment. The tightest set (unique, if it exists) with a given
13 property is contained in every other set with that property.
- 14 Also denotes one of the accuracy modes, see **tightness**.
- 15 **tightness** The strongest accuracy mode (in these are *tightest*, *accurate*, *valid*) that a given operation achieves
16 for some input box, or uniformly over some set of inputs.
- 17 NOTE—Details in 6.5.1.
- 18 **type** (or **interval type**) One of the sets into which bare and decorated interval datums are organized at
19 Level 2, usually regarded as a finite set \mathbb{T} of Level 1 intervals, (a **bare type**) in the bare case; and of Level
20 1 decorated intervals together with the value NaI (a **decorated type**).
- 21 If \mathbb{T} is a type, a **\mathbb{T} -datum** means a member of \mathbb{T} . A **\mathbb{T} -interval** for bare interval types means the same as a
22 \mathbb{T} -datum, and for decorated types means a non-NaI member of \mathbb{T} .
- 23 NOTE—Details in 6.1.

PART 2

1 **Interval Standard (Simplified)**

2 **4. Level 1 description**

3 In this clause, subclauses 4.1 to 4.4 describe the theory of set-based intervals and interval functions. Sub-
4 clause 4.5 lists the required *arithmetic operations* (also called elementary functions) with their mathematical
5 specifications.

6 **4.1 Level 1 entities**

7 Set-based intervals deal with entities of the following kinds.

- 8 – The set $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ of **extended reals**.
- 9 – The set of **(text) strings**, namely finite sequences of **characters** chosen from some alphabet.
- 10 – The set of **integers**.
- 11 – The **boolean** values **false** and **true**.
- 12 – The set of **decorations** (defined in 5).

13 Any member of $\overline{\mathbb{R}}$ is called a number. It is a **finite number** if it belongs to \mathbb{R} , else an **infinite number**.
14 An interval's members are finite numbers, but its bounds can be infinite. Finite or infinite numbers can be
15 inputs to interval constructors, as well as outputs from operations, e.g., the interval width operation.

16 Since Level 1 is primarily for human communication, there are no Level 1 restrictions on the alphabet used.
17 Strings may be inputs to interval constructors, as well as inputs/outputs of read/write operations.

18 **4.2 Intervals**

The set of mathematical intervals is denoted by $\overline{\mathbb{IR}}$. It consists of exactly those subsets \mathbf{x} of the real line
 \mathbb{R} that are closed and connected in the topological sense. Thus, it comprises the empty set (denoted \emptyset or
Empty) together with all the nonempty intervals, denoted $[\underline{x}, \overline{x}]$ and defined by

$$[\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}, \quad (1)$$

19 where \underline{x} and \overline{x} , the **bounds** of the interval, are extended-real numbers satisfying $\underline{x} \leq \overline{x}$, $\underline{x} < +\infty$ and
20 $\overline{x} > -\infty$.

21 This definition implies $-\infty$ and $+\infty$ can be bounds of an interval, but are never members of it. In particular,
22 $[-\infty, +\infty]$ is the set of all *real* numbers satisfying $-\infty \leq x \leq +\infty$, which is the whole real line \mathbb{R} —not the
23 whole extended real line $\overline{\mathbb{R}}$. Another name for the whole real line is Entire.

24 NOTE 1—The set of intervals $\overline{\mathbb{IR}}$ could be described more concisely as comprising all sets $\{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$ for
25 *arbitrary* extended-real $\underline{x}, \overline{x}$. However, this obtains Empty in many ways, as $[\underline{x}, \overline{x}]$ for any bounds satisfying $\underline{x} > \overline{x}$,
26 and also as $[-\infty, -\infty]$ or $[+\infty, +\infty]$. The description (1) was preferred as it makes a one-to-one mapping between valid
27 pairs $\underline{x}, \overline{x}$ of bounds and the nonempty intervals they specify.

28 A **box** or **interval vector** is an n -tuple $(\mathbf{x}_1, \dots, \mathbf{x}_n)$, whose components $\mathbf{x}_i \in \overline{\mathbb{IR}}$. The box \mathbf{x} is empty if
29 (and only if) any of its components \mathbf{x}_i is empty.

1 4.3 Hull

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

5 4.4 Functions

6 4.4.1 Function terminology

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

9 – A *point function* (4.4.2) is a partial mathematical real function of real variables. Otherwise, *function* is
10 usually used with its general mathematical meaning.

11 – An *arithmetic operation* (4.4.3) is a point function for which an implementation provides versions in the
12 implementation's *library* (4.4.3).

13 – A *version* of a point function f means a function derived from f ; typically a bare or decorated interval
14 extension (4.4.4) of f .

15 – An *interval arithmetic operation* is an interval extension of a point arithmetic operation (4.4.4).

16 – An *interval non-arithmetic operation* is an interval-to-interval library function that is not an interval
17 arithmetic operation (4.4.4).

18 – A *constructor* is a function that creates an interval from non-interval data (4.5.5).

19 4.4.2 Point function

20 A **point function** is a (possibly partial) multivariate real function: that is, a mapping f from a subset D of
21 \mathbb{R}^n to \mathbb{R}^m for some integers $n \geq 0, m > 0$. It is a *scalar* function if $m = 1$, otherwise a *vector* function. When
22 not otherwise specified, scalar is assumed.

The set D where f is defined is its **domain**, also written $\text{Dom } f$. To specify n , call f an n -variable point
function, or denote values of f as

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

The **range** of f over an arbitrary subset s of \mathbb{R}^n is the set $\text{Rge}(f | s)$ defined by

$$\text{Rge}(f | s) = \{ f(x) \mid x \in s \text{ and } x \in \text{Dom } f \}.$$

23 Thus mathematically, when evaluating a function over a set, points outside the domain are ignored—e.g.,
24 $\text{Rge}(\text{sqrt} | [-1, 1]) = [0, 1]$.

25 Equivalently, for the case where f takes separate arguments s_1, \dots, s_n , each being a subset of \mathbb{R} , the range
26 is written as $\text{Rge}(f | s_1, \dots, s_n)$.

27 4.4.3 Point arithmetic operation

28 A (point) **arithmetic operation** is a function for which an implementation provides versions in a collection
29 of user-available operations called its **library**. This includes functions normally written in operator form
30 (e.g., $+$, \times) and those normally written in function form (e.g., exp , arctan). It is not specified (at Level 1)
31 how an implementation provides library facilities.

1 **4.4.4 Interval-valued functions**

2 Let f be an n -variable scalar point function. An **interval extension** of f is a (total) mapping \mathbf{f} from
 3 n -dimensional boxes to intervals, that is $\mathbf{f} : \overline{\mathbb{R}}^n \rightarrow \overline{\mathbb{R}}$, such that $f(x) \in \mathbf{f}(\mathbf{x})$ whenever $x \in \mathbf{x}$ and $f(x)$ is
 4 defined, equivalently

$$\mathbf{f}(\mathbf{x}) \supseteq \text{Rge}(f | \mathbf{x})$$

5 for any box $\mathbf{x} \in \overline{\mathbb{R}}^n$, regarded as a subset of \mathbb{R}^n .

6 The **tightest interval extension** of f is the mapping \mathbf{f} defined by

$$\mathbf{f}(\mathbf{x}) = \text{hull}(\text{Rge}(f | \mathbf{x})).$$

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

$$\mathbf{f}(\mathbf{x}_1, \dots, \mathbf{x}_n) \supseteq \text{Rge}(f | \mathbf{x}_1, \dots, \mathbf{x}_n),$$

8 and the tightest interval extension is defined by

$$\mathbf{f}(\mathbf{x}_1, \dots, \mathbf{x}_n) = \text{hull}(\text{Rge}(f | \mathbf{x}_1, \dots, \mathbf{x}_n))$$

9 for any intervals $\mathbf{x}_1, \dots, \mathbf{x}_n$.

10 When f is a binary operator \bullet written in infix notation, its tightest interval extension is

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

11 [Example. With these definitions, the relevant tightest extensions satisfy $\sqrt{[-1, 4]} = [0, 2]$ and $\sqrt{[-2, -1]} = \emptyset$; also
 12 $\mathbf{x} \times [0, 0] = [0, 0]$ for any nonempty \mathbf{x} , and $\mathbf{x}/[0, 0] = \emptyset$, for any \mathbf{x} .]

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

16 An interval-valued function in the library is called an

17 – **interval arithmetic operation**, if it is an interval extension of a point arithmetic operation, and an

18 – **interval non-arithmetic operation** otherwise.

19 Examples of the latter are interval intersection and convex hull, $(\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x} \cap \mathbf{y}$ and $(\mathbf{x}, \mathbf{y}) \mapsto \text{hull}(\mathbf{x} \cup \mathbf{y})$.

20 **4.4.5 Constants**

21 A real scalar function with no arguments—a mapping $\mathbb{R}^n \rightarrow \mathbb{R}^m$ with $n = 0$ and $m = 1$ —is a **real constant**.

22 An interval extension of a real constant is any zero-argument interval function that returns an interval
 23 containing c . The *tightest extension* returns the interval $[c, c]$.

24 **4.5 Required operations**

25 **4.5.1 Interval constants**

26 The constant functions `empty()` and `entire()` have value `Empty` and `Entire` respectively.

27 **4.5.2 Arithmetic operations**

28 Table 4.1 lists required arithmetic operations, including those normally written in function notation $f(x, y, \dots)$
 29 and those normally written in unary or binary operator notation, $\bullet x$ or $x \bullet y$.

30 Each one is continuous at each point of its domain, except where stated in the footnotes after this table.
 31 Square and round brackets are used to include or exclude an interval bound, e.g., $(-\pi, \pi]$ denotes $\{x \in \mathbb{R} \mid$
 32 $-\pi < x \leq \pi\}$.

33

Table 4.1. Required forward elementary functions.

Name	Definition	Point function domain	Point function range	Table Footnotes
<i>Basic operations</i>				
neg(x)	$-x$	\mathbb{R}	\mathbb{R}	
add(x, y)	$x + y$	\mathbb{R}^2	\mathbb{R}	
sub(x, y)	$x - y$	\mathbb{R}^2	\mathbb{R}	
mul(x, y)	xy	\mathbb{R}^2	\mathbb{R}	
div(x, y)	x/y	$\mathbb{R}^2 \setminus \{y = 0\}$	\mathbb{R}	a
recip(x)	$1/x$	$\mathbb{R} \setminus \{0\}$	$\mathbb{R} \setminus \{0\}$	
sqr(x)	x^2	\mathbb{R}	$[0, \infty)$	
sqrt(x)	\sqrt{x}	$[0, \infty)$	$[0, \infty)$	
fma(x, y, z)	$(x \times y) + z$	\mathbb{R}^3	\mathbb{R}	
<i>Power functions</i>				
pown(x, p)	$x^p, p \in \mathbb{Z}$	$\begin{cases} \mathbb{R} & \text{if } p \geq 0 \\ \mathbb{R} \setminus \{0\} & \text{if } p < 0 \end{cases}$	$\begin{cases} \mathbb{R} & \text{if } p > 0 \text{ odd} \\ [0, \infty) & \text{if } p > 0 \text{ even} \\ \{1\} & \text{if } p = 0 \\ \mathbb{R} \setminus \{0\} & \text{if } p < 0 \text{ odd} \\ (0, \infty) & \text{if } p < 0 \text{ even} \end{cases}$	b
pow(x, y)	x^y	$\{x > 0\} \cup \{x = 0, y > 0\}$	$[0, \infty)$	a, c
exp, exp2, exp10(x)	b^x	\mathbb{R}	$(0, \infty)$	d
log, log2, log10(x)	$\log_b x$	$(0, \infty)$	\mathbb{R}	d
<i>Trigonometric/hyperbolic</i>				
sin(x)		\mathbb{R}	$[-1, 1]$	
cos(x)		\mathbb{R}	$[-1, 1]$	
tan(x)		$\mathbb{R} \setminus \{(k + \frac{1}{2})\pi k \in \mathbb{Z}\}$	\mathbb{R}	
asin(x)		$[-1, 1]$	$[-\pi/2, \pi/2]$	e
acos(x)		$[-1, 1]$	$[0, \pi]$	e
atan(x)		\mathbb{R}	$(-\pi/2, \pi/2)$	e
atan2(y, x)		$\mathbb{R}^2 \setminus \{(0, 0)\}$	$(-\pi, \pi]$	e, f, g
sinh(x)		\mathbb{R}	\mathbb{R}	
cosh(x)		\mathbb{R}	$[1, \infty)$	
tanh(x)		\mathbb{R}	$(-1, 1)$	
asinh(x)		\mathbb{R}	\mathbb{R}	
acosh(x)		$[1, \infty)$	$[0, \infty)$	
atanh(x)		$(-1, 1)$	\mathbb{R}	
<i>Integer functions</i>				
sign(x)		\mathbb{R}	$\{-1, 0, 1\}$	h
ceil(x)		\mathbb{R}	\mathbb{Z}	i
floor(x)		\mathbb{R}	\mathbb{Z}	i
trunc(x)		\mathbb{R}	\mathbb{Z}	i
roundTiesToEven(x)		\mathbb{R}	\mathbb{Z}	j
roundTiesToAway(x)		\mathbb{R}	\mathbb{Z}	j
<i>Absmax functions</i>				
abs(x)	$ x $	\mathbb{R}	$[0, \infty)$	
min(x, y)		\mathbb{R}^2	\mathbb{R}	k
max(x, y)		\mathbb{R}^2	\mathbb{R}	k

Footnotes to Table 4.1

- a. In describing the domain, notation such as $\{y = 0\}$ is short for $\{(x, y) \in \mathbb{R}^2 \mid y = 0\}$, etc.
- b. Regarded as a family of functions of one real variable x , parameterized by the integer argument p .
- c. Defined as $e^{y \ln x}$ for real $x > 0$ and all real y , and 0 for $x = 0$ and $y > 0$, else has no value. It is continuous at each point of its domain, including the positive y axis which is on the boundary of the domain.
- d. $b = e, 2$ or 10 , respectively.
- e. The ranges shown are the mathematical range of the point function. To ensure containment, an interval result may include values outside the mathematical range.
- f. $\text{atan2}(y, x)$ is the principal value of the argument (polar angle) of (x, y) in the plane. It is discontinuous on the half-line $y = 0, x < 0$ contained within its domain.
- g. To avoid confusion with notation for open intervals, in this table coordinates in \mathbb{R}^2 are delimited by angle brackets $\langle \rangle$.
- h. $\text{sign}(x)$ is -1 if $x < 0$; 0 if $x = 0$; and 1 if $x > 0$. It is discontinuous at 0 in its domain.
- i. $\text{ceil}(x)$ is the smallest integer $\geq x$. $\text{floor}(x)$ is the largest integer $\leq x$. $\text{trunc}(x)$ is the nearest integer to x in the direction of zero. ceil and floor are discontinuous at each integer. trunc is discontinuous at each nonzero integer.
- j. $\text{roundTiesToEven}(x), \text{roundTiesToAway}(x)$ are the nearest integer to x , with ties rounded to the even integer or away from zero, respectively. They are discontinuous at each $x = n + \frac{1}{2}$ where n is an integer.
- k. Smallest, or largest, of its real arguments.

2 4.5.3 Cancellative addition and subtraction

3 For intervals \mathbf{x} and \mathbf{y} , cancellative subtraction, $\text{cancelMinus}(\mathbf{x}, \mathbf{y})$, determines the tightest interval \mathbf{z} such
4 that

$$\mathbf{y} + \mathbf{z} \supseteq \mathbf{x},$$

5 if such a \mathbf{z} exists.

6 At Level 1,

$$\mathbf{z} = \begin{cases} \emptyset & \text{if } \mathbf{x} = \emptyset \text{ and } \mathbf{y} \text{ is bounded} \\ [\underline{x} - \underline{y}, \bar{x} - \bar{y}] & \text{if } \mathbf{x} \text{ and } \mathbf{y} \text{ are both nonempty and bounded and } \bar{y} - \underline{y} \leq \bar{x} - \underline{x}, \end{cases} \quad (2)$$

7 and \mathbf{z} has no value in the cases when

$$\begin{cases} \text{either of } \mathbf{x} \text{ or } \mathbf{y} \text{ is unbounded, or} \\ \mathbf{x} \neq \emptyset \text{ and } \mathbf{y} = \emptyset, \text{ or,} \\ \mathbf{x} \text{ and } \mathbf{y} \text{ are both nonempty and bounded and } \bar{y} - \underline{y} > \bar{x} - \underline{x}. \end{cases} \quad (3)$$

8 The operation $\text{cancelPlus}(\mathbf{x}, \mathbf{y})$ is equivalent to $\text{cancelMinus}(\mathbf{x}, -\mathbf{y})$, and therefore not considered sepa-
9 rately.

4.5.4 Set operations The intersection and convex hull operations shall be provided as in Table 4.2.

Table 4.2. Set operations.

Name	Value
$\text{intersection}(\mathbf{a}, \mathbf{b})$	intersection $\mathbf{a} \cap \mathbf{b}$ of the intervals \mathbf{a} and \mathbf{b}
$\text{convexHull}(\mathbf{a}, \mathbf{b})$	interval hull of the union $\mathbf{a} \cup \mathbf{b}$ of the intervals \mathbf{a} and \mathbf{b}

10

11 4.5.5 Constructors

An interval constructor is an operation that creates a bare or decorated interval from non-interval data. The constructors numsToInterval and textToInterval shall be provided with values as defined below:

$$\text{numsToInterval}(l, u) = \begin{cases} [l, u] = \{x \in \mathbb{R} \mid l \leq x \leq u\} & \text{if } l \leq u, l < +\infty \text{ and } u > -\infty \\ \text{no value} & \text{otherwise,} \end{cases}$$

12 where l and u are extended-real values; and

$$\text{textToInterval}(s) = \begin{cases} \text{interval denoted by } \mathbf{s} & \text{if } \mathbf{s} \text{ is a valid interval literal (see 6.6)} \\ \text{no value} & \text{otherwise.} \end{cases}$$

1 **4.5.6 Numeric functions of intervals**

- 2 The operations in Table 4.3 shall be provided, the argument being an interval and the result a number, which
3 for some of the operations may be infinite.
- 4 Implementations should provide an operation that returns $\text{mid}(\mathbf{x})$ and $\text{rad}(\mathbf{x})$ simultaneously.

Table 4.3. Required numeric functions of an interval $\mathbf{x} = [\underline{x}, \bar{x}]$.

Note: inf can have value $-\infty$; each of sup , wid , rad and mag can have value $+\infty$.

Name	Definition
$\text{inf}(\mathbf{x})$	$\begin{cases} \text{lower bound of } \mathbf{x}, \text{ if } \mathbf{x} \text{ is nonempty} \\ +\infty, \text{ if } \mathbf{x} \text{ is empty} \end{cases}$
$\text{sup}(\mathbf{x})$	$\begin{cases} \text{upper bound of } \mathbf{x}, \text{ if } \mathbf{x} \text{ is nonempty} \\ -\infty, \text{ if } \mathbf{x} \text{ is empty} \end{cases}$
$\text{mid}(\mathbf{x})$	$\begin{cases} \text{midpoint } (\underline{x} + \bar{x})/2, \text{ if } \mathbf{x} \text{ is nonempty bounded} \\ \text{no value, if } \mathbf{x} \text{ is empty or unbounded} \end{cases}$
$\text{wid}(\mathbf{x})$	$\begin{cases} \text{width } \bar{x} - \underline{x}, \text{ if } \mathbf{x} \text{ is nonempty} \\ \text{no value, if } \mathbf{x} \text{ is empty} \end{cases}$
$\text{rad}(\mathbf{x})$	$\begin{cases} \text{radius } (\bar{x} - \underline{x})/2, \text{ if } \mathbf{x} \text{ is nonempty} \\ \text{no value, if } \mathbf{x} \text{ is empty} \end{cases}$
$\text{mag}(\mathbf{x})$	$\begin{cases} \text{magnitude } \sup\{ x \mid x \in \mathbf{x}\}, \text{ if } \mathbf{x} \text{ is nonempty} \\ \text{no value, if } \mathbf{x} \text{ is empty} \end{cases}$
$\text{mig}(\mathbf{x})$	$\begin{cases} \text{mignitude } \inf\{ x \mid x \in \mathbf{x}\}, \text{ if } \mathbf{x} \text{ is nonempty} \\ \text{no value, if } \mathbf{x} \text{ is empty} \end{cases}$

5 **4.5.7 Boolean functions of intervals**

- 6 The six boolean functions in Tables 4.4 and 4.5 shall be provided. In this document, the boolean values **true**
7 and **false** are given numeric values 1 and 0 respectively, but a language may have a different mapping of
8 booleans to numbers, or no such mapping.

Table 4.4. The `isEmpty` and `isEntire` functions.

Name	Returns
<code>isEmpty(\mathbf{x})</code>	1 if \mathbf{x} is the empty set, 0 otherwise
<code>isEntire(\mathbf{x})</code>	1 if \mathbf{x} is the whole line, 0 otherwise

- 9 In Table 4.5, column three gives the set-theoretic definition, and column four gives an equivalent specification
10 when both intervals are nonempty. Table 4.6 shows what the definitions imply when at least one interval is
11 empty.

Table 4.5. Comparisons for intervals \mathbf{a} and \mathbf{b} . Notation \forall_a means “for all a in \mathbf{a} ”, and so on. In column 4, $\mathbf{a}=[\underline{a}, \bar{a}]$ and $\mathbf{b}=[\underline{b}, \bar{b}]$, where $\underline{a}, \underline{b}$ may be $-\infty$, and \bar{a}, \bar{b} may be $+\infty$; and $<'$ is the same as $<$ except that $-\infty <' -\infty$ and $+\infty <' +\infty$ are true.

Name	Symbol	Definition	For $\mathbf{a}, \mathbf{b} \neq \emptyset$	Description
equal(\mathbf{a}, \mathbf{b})	$\mathbf{a} = \mathbf{b}$	$\forall_a \exists_b a = b \wedge \forall_b \exists_a b = a$	$\underline{a} = \underline{b} \wedge \bar{a} = \bar{b}$	\mathbf{a} equals \mathbf{b}
subset(\mathbf{a}, \mathbf{b})	$\mathbf{a} \subseteq \mathbf{b}$	$\forall_a \exists_b a = b$	$\underline{b} \leq \underline{a} \wedge \bar{a} \leq \bar{b}$	\mathbf{a} is a subset of \mathbf{b}
interior(\mathbf{a}, \mathbf{b})	$\mathbf{a} \textcircled{a} \mathbf{b}$	$\forall_a \exists_b a < b \wedge \forall_a \exists_b b < a$	$\underline{b} <' \underline{a} \wedge \bar{a} <' \bar{b}$	\mathbf{a} is interior to \mathbf{b}
disjoint(\mathbf{a}, \mathbf{b})	$\mathbf{a} \not\cap \mathbf{b}$	$\forall_a \forall_b a \neq b$	$\bar{a} < \underline{b} \vee \bar{b} < \underline{a}$	\mathbf{a} and \mathbf{b} are disjoint

Table 4.6. Comparisons with empty intervals.

	$\mathbf{a} = \emptyset$	$\mathbf{a} \neq \emptyset$	$\mathbf{a} = \emptyset$
	$\mathbf{b} \neq \emptyset$	$\mathbf{b} = \emptyset$	$\mathbf{b} = \emptyset$
$\mathbf{a} = \mathbf{b}$	0	0	1
$\mathbf{a} \subseteq \mathbf{b}$	1	0	1
$\mathbf{a} \textcircled{a} \mathbf{b}$	1	0	1
$\mathbf{a} \not\cap \mathbf{b}$	1	1	1

1 5. The decoration system at Level 1

2 5.1 Overview

3 An implementation makes the decoration system available by providing:

- 4 – a decorated version of each interval extension of an arithmetic operation, of each interval constructor, and
- 5 of some other operations; and
- 6 – various auxiliary functions, e.g., to extract the interval and decoration parts, and to apply a standard
- 7 initial decoration to an interval.

8 The decoration system is specified here at a mathematical level, with the finite-precision aspects presented
9 in Clause 6. Subclauses 5.2, 5.3, and 5.4 give the basic concepts; 5.5 and 5.6 define how intervals are given
10 an initial decoration, and how decorations are bound to library interval arithmetic operations to give correct
11 propagation through expressions; 5.7 is about non-arithmetic operations.

12 5.2 Definitions and properties

13 A decoration d is a property (that is, a boolean-valued function) $p_d(f, \mathbf{x})$ of pairs (f, \mathbf{x}) , where f is a real-
14 valued function with domain $\text{Dom}(f) \subseteq \mathbb{R}^n$ for some $n \geq 0$ and $\mathbf{x} \in \overline{\mathbb{R}}^n$ is an n -dimensional box, regarded
15 as a subset of \mathbb{R}^n .

The notation (f, \mathbf{x}) unless said otherwise denotes such a pair, for arbitrary n , f and \mathbf{x} . Equivalently, d is identified with the set of pairs for which the property holds:

$$d = \{ (f, \mathbf{x}) \mid p_d(f, \mathbf{x}) \text{ is true} \}. \quad (4)$$

The set \mathbb{D} of decorations has five members:

Value	Short description	Property	Definition
com	common	$p_{\text{com}}(f, \mathbf{x})$	\mathbf{x} is a bounded, nonempty subset of $\text{Dom}(f)$; f is continuous at each point of \mathbf{x} ; and the computed interval $f(\mathbf{x})$ is bounded.
dac	defined & continuous	$p_{\text{dac}}(f, \mathbf{x})$	\mathbf{x} is a nonempty subset of $\text{Dom}(f)$, and the restriction of f to \mathbf{x} is continuous. (5)
def	defined	$p_{\text{def}}(f, \mathbf{x})$	\mathbf{x} is a nonempty subset of $\text{Dom}(f)$.
trv	trivial	$p_{\text{trv}}(f, \mathbf{x})$	always true (so gives no information).
ill	ill-formed	$p_{\text{ill}}(f, \mathbf{x})$	Not an Interval; formally $\text{Dom}(f) = \emptyset$, see 5.3.

16 These are listed according to the propagation order (10), which may also be thought of as a quality-order of
17 (f, \mathbf{x}) pairs—decorations above **trv** are “good” and **ill** is “bad”.

18 A **decorated interval** is a pair, written interchangeably as (\mathbf{u}, d) or \mathbf{u}_d , where $\mathbf{u} \in \overline{\mathbb{R}}$ is a real interval and
19 $d \in \mathbb{D}$ is a decoration. (\mathbf{u}, d) may also denote a decorated box $((\mathbf{u}_1, d_1), \dots, (\mathbf{u}_n, d_n))$, where \mathbf{u} and d are the
20 vectors of interval parts \mathbf{u}_i and decoration parts d_i , respectively.

21 The set of decorated intervals is denoted by $\overline{\mathbb{D}\mathbb{I}\mathbb{R}}$, and the set of decorated boxes with n components is
22 denoted by $\overline{\mathbb{D}\mathbb{I}\mathbb{R}}^n$.

23 When several named intervals are involved, the decorations attached to $\mathbf{u}, \mathbf{v}, \dots$ are often named du, dv, \dots
24 for readability, for instance (\mathbf{u}, du) or \mathbf{u}_{du} , etc.

25 An interval may be called a **bare** interval to emphasize that it is not a decorated interval.

Treating the decorations as sets as in (4), **trv** is the set of all (f, \mathbf{x}) pairs, and the others are nonempty subsets of **trv**. By design, they satisfy the **exclusivity rule**

$$\text{For any two decorations, either one contains the other or they are disjoint.} \quad (6)$$

Namely, the definitions (5) give:

$$\text{com} \subset \text{dac} \subset \text{def} \subset \text{trv} \supset \text{ill}, \quad \text{note the change from } \subset \text{ to } \supset \quad (7)$$

$$\text{com}, \text{dac} \text{ and } \text{def} \text{ are disjoint from } \text{ill}. \quad (8)$$

Property (6) implies that for any (f, \mathbf{x}) there is a unique tightest (in the containment order (7)) decoration, such that $p_d(f, \mathbf{x})$ is true, called the **strongest decoration of (f, \mathbf{x})** , or of f over \mathbf{x} , and written $\text{Dec}(f | \mathbf{x})$. That is,

$$\text{Dec}(f | \mathbf{x}) = d \iff p_d(f, \mathbf{x}) \text{ holds, but } p_e(f, \mathbf{x}) \text{ fails for all } e \subset d.$$

- 1 NOTE—Like the exact range $\text{Rge}(f | \mathbf{x})$, the strongest decoration is theoretically well-defined, but its value for a
2 particular f and \mathbf{x} may be impractically expensive to compute, or even undecidable.

3 5.3 The ill-formed interval

4 An ill-formed decorated interval is also called NaI, **Not an Interval**. Conceptually, there shall be only one
5 NaI. Its interval part has no value at Level 1.

6 The `ill` decoration results from invalid constructions and propagates unconditionally through arithmetic
7 expressions. Namely, the `ill` decoration arises as a return value of

- 8 – a constructor when it cannot construct a valid decorated interval, or of
- 9 – a library arithmetic operation if and only if one of its inputs is ill-formed.

10 Formally, `ill` may be identified with the property $\text{Dom}(f) = \emptyset$ of (f, \mathbf{x}) pairs.

11 [Example. The constructor call `numsToInterval(2, 1)` is invalid, so its decorated version returns NaI.]

12 Information may be stored in a NaI in an implementation-defined way (like the payload of an IEEE 754
13 floating-point NaN), and functions may be provided for a user to set and read this for diagnostic purposes.

14 An implementation may provide means for an exception to be signaled when a NaI is produced.

15 5.4 Permitted combinations

16 A decorated interval \mathbf{y}_{dy} shall always be such that

$$\mathbf{y} \supseteq \text{Rge}(f | \mathbf{x}) \text{ and } p_{dy}(f, \mathbf{x}) \text{ holds, for some } (f, \mathbf{x}).$$

17 If $dy = \text{dac}$, `def` or `com`, then by definition \mathbf{x} is nonempty, and f is everywhere defined on it, so that $\text{Rge}(f | \mathbf{x})$
18 is nonempty, implying \mathbf{y} is nonempty. Hence the decorated intervals

$$\emptyset_{\text{dac}}, \emptyset_{\text{def}}, \text{ and } \mathbf{x}_{\text{com}} \text{ if } \mathbf{x} \text{ is empty or unbounded}$$

19 are contradictory—implementations shall not produce them.

20 No other combinations are forbidden.

21 5.5 Operations on/with decorations

22 5.5.1 Initializing

A bare interval is initialized with a decoration by the operation

$$\text{newDec}(\mathbf{x}) = \mathbf{x}_d \quad \text{where} \quad d = \begin{cases} \text{com} & \text{if } \mathbf{x} \text{ is nonempty and bounded,} \\ \text{dac} & \text{if } \mathbf{x} \text{ is unbounded, and} \\ \text{trv} & \text{if } \mathbf{x} \text{ is empty.} \end{cases}$$

23

1 5.5.2 Disassembling and assembling

2 For a decorated interval \mathbf{x}_{dx} , the operations `intervalPart(\mathbf{x}_{dx})` and `decorationPart(\mathbf{x}_{dx})` shall be provided,
 3 with value \mathbf{x} and dx , respectively. For the case of NaI, `decorationPart(NaI)` has the value `ill`, but
 4 `intervalPart(NaI)` has no value at Level 1.

5 Given an interval \mathbf{x} and a decoration dx , the operation `setDec(\mathbf{x}, dx)` returns the decorated interval \mathbf{x}_{dx} if
 6 this is an allowed combination. The cases of forbidden combinations are as follows:

- 7 – `setDec(\emptyset, dx)`, where dx is one of `def`, `dac` or `com`, returns \emptyset_{trv} ;
- 8 – `setDec(\mathbf{x}, com)`, for any unbounded \mathbf{x} , returns \mathbf{x}_{dac} ; and
- 9 – `setDec(\mathbf{x}, ill)` for any \mathbf{x} , whether empty or not, returns NaI.

10 5.5.3 Comparisons

11 For decorations, comparison operations for equality `=` and its negation `≠` shall be provided, as well as
 12 comparisons `>`, `<`, `≥`, `≤` with respect to the propagation order (10).

13 5.6 Decorations and arithmetic operations

14 Given a scalar point function φ of k variables, a **decorated interval extension** of φ —denoted here by the
 15 same name φ —adds a decoration component to a bare interval extension of φ . It has the form $\mathbf{w}_{dw} = \varphi(\mathbf{v}_{dv})$,
 16 where $\mathbf{v}_{dv} = (\mathbf{v}, dv)$ is a k -component decorated box $((\mathbf{v}_1, dv_1), \dots, (\mathbf{v}_k, dv_k))$. By the definition of a bare
 17 interval extension, the interval part \mathbf{w} depends only on the input intervals \mathbf{v} ; the decoration part dw generally
 18 depends on both \mathbf{v} and dv . In this context, NaI is regarded as being \emptyset_{ill} .

The definition of a bare interval extension implies

$$\mathbf{w} \supseteq \text{Rge}(\varphi | \mathbf{v}) \quad (\text{enclosure}).$$

The decorated interval extension of φ determines a dv_0 such that

$$p_{dv_0}(\varphi, \mathbf{v}) \text{ holds} \quad (\text{a “local decoration”}). \quad (9)$$

It then evaluates the output decoration dw by

$$dw = \min\{dv_0, dv_1, \dots, dv_k\}, \quad (\text{the “min-rule”}),$$

where the minimum is taken with respect to the **propagation order**:

$$\text{com} > \text{dac} > \text{def} > \text{trv} > \text{ill}. \quad (10)$$

19 5.7 Decoration of non-arithmetic operations

20 5.7.1 Interval-valued operations

21 These give interval results but are not interval extensions of point functions.

- 22 – the cancellative operations `cancelPlus(\mathbf{x}, \mathbf{y})` and `cancelMinus(\mathbf{x}, \mathbf{y})` of 4.5.3;
- 23 – the set-oriented operations `intersection(\mathbf{x}, \mathbf{y})` and `convexHull(\mathbf{x}, \mathbf{y})` of 4.5.4

24 No one way of decorating these operations gives useful information in all contexts. Therefore, a *trivial*
 25 decorated interval version is provided as follows. If any input is NaI, the result is NaI; otherwise the
 26 corresponding operation is applied to the interval parts of the inputs, and its result decorated with `trv`.
 27 The user may replace this by an appropriate nontrivial decoration via `setDec()`, see 5.5, where this can be
 28 deduced in a given application.

1 **5.7.2 Non-interval-valued operations**

2 These give non-interval results:

3 – the numeric functions of 4.5.6 and

4 – the boolean-valued functions of 4.5.7.

5 For each such operation, if any input is NaI, the result has no value at Level 1. Otherwise, the opera-
6 tion acts on decorated intervals by discarding the decoration and applying the corresponding bare interval
7 operation.

1 6. Level 2 description

2 6.1 Introduction

3 Entities and operations at Level 2 are said to have **finite precision**. From them, implementable interval
4 algorithms may be constructed. Level 2 entities are called **datums**⁴.

5 6.1.1 Interval type

6 The interval type, denoted by \mathbb{T} , is the **inf-sup type** derived from the IEEE 754 **binary64** format; we refer
7 to the latter as **b64**. This interval type comprises all intervals whose endpoints are **b64** numbers, together
8 with **Empty**. Since $\pm\infty$ are in **b64**, **Entire** is in \mathbb{T} .

9 An interval from \mathbb{T} is also called a **bare interval** or a **\mathbb{T} -interval**. We use the term **\mathbb{T} -datum** to refer
10 to an entity that can be a bare or decorated interval, including **NaN**. A **\mathbb{T} -box** is a vector with \mathbb{T} -datum
11 components.

12 6.1.2 Decorated interval type

13 The decorated interval type, derived from \mathbb{T} , is the set of tuples (x, d) , where $x \in \mathbb{T}$, and $d \in \mathbb{D}$. We denote
14 this type by \mathbb{DT} .

15 \mathbb{DT} shall contain a “Not an Interval” datum **NaN**, which is identified with $(\emptyset, \text{i11})$.

16 6.1.3 Operations

17 The term **\mathbb{T} -version** of a Level 1 operation denotes one in which any input or output that is an interval is a
18 \mathbb{T} -datum. For bare interval types this includes the following.

- 19 – An interval extension (see 6.4) of one of the arithmetic operations of 4.5.
- 20 – A set operation, such as intersection and convex hull of \mathbb{T} -intervals, returning a \mathbb{T} -interval.
- 21 – A function such as the midpoint, whose input is a \mathbb{T} -interval and output is numeric.
- 22 – A constructor, whose input is numeric or text and output is a \mathbb{T} -datum.

23 6.1.4 Exception behavior

24 For some operations, and some particular inputs, there might not be a valid result. At Level 1 there are
25 several cases when no value exists. However, a Level 2 operation always returns a value. When the Level 1
26 result does not exist, the corresponding Level 2 operation returns either

- 27 – a special value indicating this event (e.g., **NaN** for most of the numeric functions in 6.7.6); or
- 28 – a value considered reasonable in practice. For example, **mid(Entire)** returns 0; a constructor given invalid
29 input returns **Empty**; and one of the comparisons of 6.7.7, if any input is **NaN**, returns **false**.

30 If **intervalPart()** is called with **NaN** as input, the exception **IntvlPartOfNaN** is signaled (see 6.7.8).

31 If a bare or decorated constructor fails (see 6.7.5) the exception **UndefinedOperation** is signaled.

32 If a bare or decorated constructor succeeds on an accuracy-relaxed string argument (see 6.7.5) the exception
33 **PossiblyUndefinedOperation** may be signaled.

34 6.2 Naming conventions for operations

35 An operation is generally given a name that suits the context. For example, the addition of two interval
36 datums \mathbf{x}, \mathbf{y} may be written in generic algebra notation $\mathbf{x} + \mathbf{y}$; or with a generic text name **add(\mathbf{x}, \mathbf{y})**.

⁴Not “data”, whose common meaning could cause confusion.

1 **6.3 Level 2 hull operation**

2 **6.3.1 Hull in one dimension**

3 The **interval hull** operation

$$\mathbf{y} = \text{hull}(\mathbf{s}),$$

4 maps an arbitrary set of reals \mathbf{s} to the tightest interval \mathbf{y} enclosing \mathbf{s} .

5 **6.3.2 Hull in n dimensions**

6 In n dimensions the hull, as defined mathematically in 6.3.1, is extended to act componentwise. That is, for
 7 an arbitrary subset \mathbf{s} of \mathbb{R}^n it is $\text{hull}(\mathbf{s}) = (\mathbf{y}_1, \dots, \mathbf{y}_n)$, where

$$\mathbf{y}_i = \text{hull}(\mathbf{s}_i),$$

8 and $\mathbf{s}_i = \{s_i \mid s \in \mathbf{s}\}$ is the projection of \mathbf{s} on the i th coordinate dimension.

9 **6.4 Level 2 interval extensions**

Let f be an n -variable scalar point function. A **\mathbb{T} -interval extension** of f , also called a **\mathbb{T} -version** of f , is a mapping \mathbf{f} from n -dimensional \mathbb{T} -boxes to \mathbb{T} -intervals, that is $\mathbf{f} : \mathbb{T}^n \rightarrow \mathbb{T}$, such that $f(x) \in \mathbf{f}(\mathbf{x})$ whenever $x \in \mathbf{x}$ and $f(x)$ is defined. Equivalently

$$\mathbf{f}(\mathbf{x}) \supseteq \text{Rge}(f \mid \mathbf{x}),$$

10 for any \mathbb{T} -box $\mathbf{x} \in \mathbb{T}^n$, regarding \mathbf{x} as a subset of \mathbb{R}^n . Generically, such mappings are called Level 2 interval
 11 extensions.

12 **6.5 Accuracy of operations**

13 This subclause describes requirements and recommendations on the accuracy of operations. Here, *operation*
 14 denotes any Level 2 version, provided by the implementation, of a Level 1 operation with interval output and
 15 at least one interval input. Bare interval operations are described; the accuracy of a decorated operation is
 16 defined to be that of its interval part.

17 **6.5.1 Measures of accuracy**

18 Three **accuracy modes** are defined that indicate the quality of interval enclosure achieved by an operation:
 19 *tightest*, *accurate* and *valid* in order from strongest to weakest.

20 The term **tightness** means the strongest mode that holds uniformly for some set of evaluations. For example,
 21 for some one-argument function, an implementation might document the tightness of $\mathbf{f}(\mathbf{x})$ as being *tightest*
 22 for all \mathbf{x} contained in $[-10^{15}, 10^{15}]$ and at least *accurate* for all other \mathbf{x} .

Let $\mathbf{f}_{\text{exact}}$ denote the corresponding Level 1 operation. The weakest mode *valid* is just the property of enclosure:

$$\mathbf{f}(\mathbf{x}) \supseteq \mathbf{f}_{\text{exact}}(\mathbf{x}). \quad (11)$$

The strongest mode *tightest* is the property that $\mathbf{f}(\mathbf{x})$ equals $\mathbf{f}_{\text{tightest}}(\mathbf{x})$, the hull of the Level 1 result:

$$\mathbf{f}_{\text{tightest}}(\mathbf{x}) = \text{hull}(\mathbf{f}_{\text{exact}}(\mathbf{x})). \quad (12)$$

The intermediate mode *accurate* asserts that $\mathbf{f}(\mathbf{x})$ is *valid*, (11), and is at most slightly wider than the result of applying the *tightest* version to a slightly wider input box:

$$\mathbf{f}(\mathbf{x}) \subseteq \text{nextOut}(\mathbf{f}_{\text{tightest}}(\text{nextOut}(\text{hull}(\mathbf{x}))))). \quad (13)$$

For an interval \mathbf{x} ,

$$\text{nextOut}(\mathbf{x}) = \begin{cases} [\text{nextDown}(\underline{x}), \text{nextUp}(\bar{x})] & \text{if } \mathbf{x} = [\underline{x}, \bar{x}] \neq \emptyset, \\ \emptyset & \text{if } \mathbf{x} = \emptyset, \end{cases}$$

23 where **nextUp** and **nextDown** are equivalent to the corresponding functions in IEEE 754.

1 When x is an interval box, `nextOut` acts componentwise.

2 NOTE—In (13), the inner `nextOut()` aims to handle the problem of a function such as $\sin x$ evaluated at a very large
3 argument, where a small relative change in the input can produce a large relative change in the result. The outer
4 `nextOut()` relaxes the requirement for correct (rather than, say, faithful) rounding, which might be hard to achieve
5 for some special functions at some arguments.

6 6.5.2 Accuracy requirements

7 Following the categories of functions in Table 4.1, the accuracy of the *basic operations*, the *integer func-*
8 *tions* and the *absmax functions* shall be *tightest*. The accuracy of the *cancellative addition and subtraction*
9 operations of 4.5.3 is specified in 6.7.3.

10 For all other operations in Table 4.1, the accuracy should be *accurate*.

11 6.5.3 Documentation requirements

12 An implementation shall document the tightness of each of its interval operations. This shall be done by
13 dividing the set of possible inputs into disjoint subsets (“ranges”) and stating a tightness achieved in each
14 range.

15 [Example. Sample tightness information for the `sin` function might be

Operation	Tightness	Range
sin	<i>tightest</i> <i>accurate</i>	for any $x \subseteq [-10^{15}, 10^{15}]$ for all other x .

17]

18 Each operation should be identified by a language- or implementation-defined name of the Level 1 operation
19 (which might differ from that used in this standard), its output type, its input type(s) if necessary, and any
20 other information needed to resolve ambiguity.

21 6.6 Number and interval literals

22 6.6.1 Number literals

23 The following forms of number literal shall be provided.

- 24 a) A decimal number. This comprises an optional sign, a nonempty sequence of decimal digits optionally
25 containing a point, and an optional exponent field comprising `e` and an integer literal.⁵ The value of a
26 decimal number is the value of the sequence of decimal digits with optional point multiplied by ten raised
27 to the power of the value of the integer literal, negated if there is a leading `-` sign.
- 28 b) A number in the hexadecimal-floating-constant form of the C99 standard (ISO/IEC9899, N1256 (6.4.4.2)),
29 equivalently hexadecimal-significand form of IEEE Std 754-2008 (5.12.3). This comprises an optional sign,
30 the string `0x`, a nonempty sequence of hexadecimal digits optionally containing a point, and an exponent
31 field comprising `p` and an integer literal exponent. The value of a hexadecimal number is the value of the
32 sequence of hexadecimal digits with optional point multiplied by two raised to the power of the value of
33 the exponent, negated if there is a leading minus sign.
- 34 c) A rational literal p/q . This comprises an integer literal p , the `/` character, and a positive-natural literal
35 q . Its value is the value of p divided by the value of q .
- 36 d) Either of the strings `inf` or `infinity` optionally preceded by `+`, with value $+\infty$; or preceded by `-`, with
37 value $-\infty$.

⁵An integer literal comprises an optional sign and followed by a nonempty sequence of decimal digits.

1 6.6.2 Bare intervals

2 The following forms of bare interval literal shall be supported. Below, the number literals l and r are identified
 3 with their values. Space shown between elements of a literal denotes zero or more space characters.

4 – A string $[l , u]$ where l and u are optional number literals with $l \leq u$, $l < +\infty$ and $u > -\infty$, see 4.2. Its
 5 bare value is the mathematical interval $[l, u]$. Any of l and u may be omitted, with implied values $l = -\infty$
 6 and $u = +\infty$, respectively; e.g. $[,]$ denotes Entire.

7 A string $[x]$ is equivalent to $[x , x]$.

8 – Uncertain form: a string $m ? r v E$ where: m is a decimal number literal of form a) in 6.6.1, without
 9 exponent; r is empty or is a natural-number literal *ulp-count* or is $?$; v is empty or is a *direction character*,
 10 either **u** (up) or **d** (down); and E is empty or is an *exponent field* comprising the character **e** followed by
 11 an integer literal *exponent* e . No whitespace is permitted within the string.

12 With **ulp** meaning 10^{-d} , where d is the number of digits after the decimal point in m (or 0 if there is no
 13 decimal point), the literal $m?$ by itself denotes m with a symmetrical uncertainty of half an ulp, that is
 14 the interval $[m - \frac{1}{2}\text{ulp}, m + \frac{1}{2}\text{ulp}]$. The literal $m?r$ denotes m with a symmetrical uncertainty of r ulps,
 15 that is $[m - r \times \text{ulp}, m + r \times \text{ulp}]$. Adding **d** (down) or **u** (up) converts this to uncertainty in one direction
 16 only, e.g., $m?d$ denotes $[m - \frac{1}{2}\text{ulp}, m]$ and $m?ru$ denotes $[m, m + r \times \text{ulp}]$. Uncertain form with radius $?$
 17 is for unbounded intervals, e.g., $m??d$ denotes $[-\infty, m]$. The exponent field if present multiplies the whole
 18 interval by 10^e , e.g., $m?ru ee$ denotes $10^e \times [m, m + r \times \text{ulp}]$.

19 – Special values: the strings $[]$ and $[\text{empty}]$, whose bare value is Empty, and the string $[\text{entire}]$, whose
 20 bare value is Entire.

21 6.6.3 Decorated intervals

22 The following forms of decorated interval literal shall be supported.

23 – **sx_sd** : a bare interval literal **sx** , an underscore “**_**”, and a 3-character decoration string **sd** , where **sd** is
 24 one of **trv**, **def**, **dac** or **com**, denoting the corresponding decoration dx .

25 If **sx** has the bare value x , and if x_{dx} is a permitted combination according to 5.4, then **sx_sd** has the
 26 value x_{dx} . Otherwise **sx_sd** has no value as a decorated interval literal.

27 – The string $[\text{nai}]$, with the bare value Empty and the decorated value Empty_{i11} .

28 The alphanumeric characters in the above literals are case-insensitive (e.g., $[1,1e3]_{\text{com}}$ is equivalent to
 29 $[1,1E3]_{\text{COM}}$).

30 6.7 Required operations

31 Operations in this subclause are described as functions with zero or more input arguments and one return
 32 value. It is language-defined whether they are implemented in this way.

33 6.7.1 Interval constants

There shall be functions **empty()** and **entire()** returning an interval with value Empty and Entire, respec-
 tively. There shall also be a decorated version of each, returning

$$\text{newDec}(\text{Empty}) = \text{Empty}_{\text{trv}} \quad \text{and} \quad \text{newDec}(\text{Entire}) = \text{Entire}_{\text{dac}},$$

34 respectively.

35 6.7.2 Elementary functions

36 An implementation shall provide an interval version of each arithmetic operation in Table 4.1. Its inputs
 37 and output are intervals, and it shall be a Level 2 interval extension of the corresponding point function.
 38 Recommended accuracies are given in 6.5.

39 NOTE—For operations, some of whose arguments are of integer type, such as integer power **pow** $n(x, p)$, only the real
 40 arguments are replaced by intervals.

1 Each such operation shall have a decorated version with corresponding arguments of type \mathbb{DT} . It shall be a
2 decorated interval extension as defined in 5.6—thus the interval part of its output is the same as if the bare
3 interval operation were applied to the interval parts of its inputs.

4 The only freedom of choice in the decorated version is how the local decoration, denoted dv_0 in (9) of 5.6,
5 is computed. dv_0 shall be the strongest possible (and is thus uniquely defined), if the accuracy mode of the
6 corresponding bare interval operation is “tightest”, but otherwise is only required to obey (9).

7 **6.7.3 Cancellative addition and subtraction**

8 An implementation shall provide a \mathbb{T} -version of each of the operations `cancelMinus` and `cancelPlus` in 4.5.3.
9 Their inputs and output are \mathbb{T} -intervals.

10 `cancelMinus(x, y)` shall return `Empty` in the first case of (2), the hull of the result in the second, and `Entire`
11 for each of the cases in (3).

12 `cancelPlus(x, y)` shall be equivalent to `cancelMinus(x, -y)`.

13 These operations shall have “trivial” decorated versions, as described in 5.7.

14 **6.7.4 Set operations**

15 An implementation shall provide an interval version of each of the operations `intersection` and `convexHull`
16 in 4.5.4. Its inputs and output are intervals. These operations should return the interval hull of the exact
17 result. If either input to `intersection` is `Empty`, or both inputs to `convexHull` are `Empty`, the result shall
18 be `Empty`.

19 These operations shall have “trivial” decorated versions, as described in 5.7.

20 **6.7.5 Constructors**

21 For the bare and decorated interval types there shall be a constructor. It returns a \mathbb{T} - or \mathbb{DT} -datum,
22 respectively.

23 **Bare interval constructors.** A bare interval constructor call either **succeeds** or **fails**. This notion is used to
24 determine the value returned by the corresponding decorated interval constructor.

25 For the constructor `numsToInterval(l, u)`, the inputs l and u are `b64` datums. If neither l nor u is `NaN`, and
26 $l \leq u$, $l < +\infty$, $u > -\infty$, the result is $[l, u]$. Otherwise the call fails, and the result is `Empty`.

27 For the constructor `textToInterval(s)`, the input s is a string. The string of form $[l , u]$ where $l < +\infty$
28 and $u > -\infty$ are optional number literals is called **accuracy-relaxed** if either one of the literals is a rational
29 number literal of form c) in 6.6.1 or one of them is a decimal number literal of form a) in 6.6.1 and another
30 is a hexadecimal number literal of form b) in 6.6.1.

31 – If s is a valid interval literal with Level 1 value x and s is not accuracy-relaxed, the result shall be the hull
32 of x (the constructor succeeds).

33 – If s is not a valid interval interval and s is not accuracy-relaxed, this constructor fails, and the result is
34 `Empty`.

35 – If s is accuracy-relaxed and $l \leq u$, the result may be any interval containing $[l, u]$ (the constructor
36 succeeds).

37 – If s is accuracy-relaxed and $l > u$, the result may be either any interval containing $[u, l]$ (the constructor
38 succeeds), or `Empty` (the constructor fails).

39 **Decorated interval constructors.** Let the prefix `b-` or `d-` denote the bare or decorated version of a constructor.
40 If `b-numsToInterval(l, u)` or `b-textToInterval(s)` succeeds with result y , then `d-numsToInterval(l, u)` or
41 `d-textToInterval(s)`, respectively, succeeds with result y and decoration `newDec(y)`, see 5.5.

1 If s is a decorated interval literal $sx.sd$ with Level 1 value x_{dx} , see 6.6.3, and `b-textToInterval(sx)` succeeds
 2 with result y , then `d-textToInterval(s)` succeeds with result y_{dy} , where $dy = dx$ except when $dx = \text{com}$
 3 and overflow has occurred, that is, x is bounded and y is unbounded. Then dy shall equal `dac`.

4 Otherwise the call fails, and the result is `NaN`.

5 **Exception behavior.** Exception `UndefinedOperation` is signaled by both the bare and the decorated con-
 6 structor when the input is such that the bare constructor fails.

7 Exception `PossiblyUndefinedOperation` is signaled by both the bare and the decorated `textToInterval(s)`
 8 constructor with accuracy-relaxed s

- 9 – when $l > u$ and interval constructor succeeds;
- 10 – when $l \leq u$ and the result is wider than the hull of $[l, u]$;
- 11 – optionally when $l \leq u$ and the result is the hull of $[l, u]$.

12 NOTE—When signaled by the decorated constructor it will normally be ignored since returning `NaN` gives sufficient
 13 information.

14 NOTE—If the `textToInterval(s)` constructor does not signal an exception then s is a valid interval literal and the
 15 result is the hull of Level 1 value of s .

16 NOTE—The behavior of the `textToInterval(s)` constructor is implementation-dependent when s is accuracy-relaxed.
 17 The least accurate implementation simply returns `Entire` and signals `PossiblyUndefinedOperation`. The most ac-
 18 curate implementation fails with `UndefinedOperation` exception when $l > u$ and returns the hull of $[l, u]$ without
 19 exception otherwise.

20 6.7.6 Numeric functions of intervals

21 An implementation shall provide a \mathbb{T} -version of each numeric function in Table 4.3 of 4.5.6 giving a result
 22 in `b64`. The mapping of a Level 1 value to a `b64` number is defined in terms of the following rounding
 23 methods:

24 *Round toward positive:* x maps to the smallest `b64` number not less than x ; 0 maps to `+0`.

25 *Round toward negative:* x maps to the largest `b64` number not greater than x ; 0 maps to `-0`.

26 *Round to nearest:* x maps to the `b64` number (possibly $\pm\infty$) closest to x as defined by IEEE 754-2008;
 27 0 maps to `+0`.

28 NOTE—These functions help define operations of the standard but are not themselves operations of the stan-
 29 dard.

30 A Level 1 value of 0 shall be returned as `-0` by `inf`, and `+0` by all other functions in this subclause.

31 `inf(x)` returns the Level 1 value rounded toward negative.

32 `sup(x)` returns the Level 1 value rounded toward positive.

33 `mid(x)`: the result is defined by the following cases, where \underline{x}, \bar{x} are the exact (Level 1) lower and upper
 34 bounds of x :

$x = \text{Empty}$	NaN
$x = \text{Entire}$	0
$\underline{x} = -\infty, \bar{x}$ finite	the finite negative <code>b64</code> number of largest magnitude
\underline{x} finite, $\bar{x} = +\infty$	the finite positive <code>b64</code> number of largest magnitude
\underline{x}, \bar{x} both finite	the Level 1 value rounded to nearest

35 The implementation shall document how it handles the last case.

36 `rad(x)` returns `NaN` if x is empty, and otherwise the smallest `b64` number r such that x is contained in the
 37 exact interval $[m - r, m + r]$, where m is the value returned by `mid(x)`.

- 1 `wid(x)` returns NaN if x is empty. Otherwise it returns the Level 1 value rounded toward positive.
- 2 `mag(x)` returns NaN if x is empty. Otherwise it returns the Level 1 value rounded toward positive.
- 3 `mig(x)` returns NaN if x is empty. Otherwise it returns the Level 1 value rounded toward negative, except
4 that 0 maps to +0.
- 5 Each bare interval operation in this subclause shall have a decorated version, where each input of bare interval
6 type is replaced by one of the corresponding decorated interval type, and the result format is that of the
7 bare operation. Following 5.7, if any input is NaI, the result is NaN. Otherwise the result is obtained by
8 discarding the decoration and applying the corresponding bare interval operation.

9 **6.7.7 Boolean functions of intervals**

10 An implementation shall provide the functions

- 11 – `isEmpty(x)` and `isEntire(x)` in 4.5.7,
12 – the functions implementing the comparison relations in Table 4.5 of 4.5.7, and
13 – the function `isNaI(x)` for input x of the decorated type, which returns `true` if x is NaI, else `false`.

14 Each bare interval operation in this subclause shall have a decorated version. Following 5.7, if any input
15 is NaI, the result is `false` (in particular `equal(NaI,NaI)` is `false`). Otherwise the result is obtained by
16 discarding the decoration and applying the corresponding bare interval operation.

17 **6.7.8 Operations on/with decorations**

18 An implementation shall provide the operations of 5.5. These comprise the comparison operations `=`, `≠`, `>`,
19 `<`, `≥`, `≤` for decorations; and, for the decorated type, the operations `newDec`, `intervalPart`, `decorationPart`
20 and `setDec`.

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

1 **6.8 Input and output (I/O) of intervals**

2 **6.8.1 Overview**

3 This clause specifies conversion from a text string that holds an interval literal to an interval internal to a
4 program (input), and the reverse (output). The methods by which strings are read from, or written to, a
5 character stream are language- or implementation-defined, as are variations in some locales (such as specific
6 character case matching).

7 Containment is preserved on input and output so that, when a program computes an enclosure of some quan-
8 tity given an enclosure of the data, it can ensure this holds all the way from text data to text results.

9 **6.8.2 Input**

10 Input is provided by `textToInterval(s)`; see 6.7.5.

11 **6.8.3 Output**

12 An implementation shall provide an operation

$$\text{intervalToText}(\mathbf{X}, cs)$$

13 where cs is optional. \mathbf{X} is a bare or decorated interval, and cs is a string, the conversion specifier. This
14 operation converts \mathbf{X} to a valid interval literal string s , see 6.7.1, which shall be related to \mathbf{X} as follows,
15 where \mathbf{Y} below is the Level 1 value of s .

16 a) Let \mathbf{X} be a bare interval. Then \mathbf{Y} shall contain \mathbf{X} and shall be empty if \mathbf{X} is empty.

17 b) Let \mathbf{X} be a decorated interval. If \mathbf{X} is NaI then \mathbf{Y} shall be NaI. Otherwise, write $\mathbf{X} = \mathbf{x}_{dx}$, $\mathbf{Y} = \mathbf{y}_{dy}$.
18 Then

19 1) \mathbf{y} shall contain \mathbf{x} , and shall be empty if \mathbf{x} is empty.

20 2) dy shall equal dx , except in the case that $dx = \text{com}$ and overflow occurred, that is, \mathbf{x} is bounded and
21 \mathbf{y} is unbounded. Then dy shall equal dac .

22 The tightness of enclosure of \mathbf{X} by the value \mathbf{Y} of s is language- or implementation-defined.

23 If present, cs lets the user control the layout of the string s in a language- or implementation-defined way.
24 The implementation shall document the recognized values of cs and their effect; other values are called
25 *invalid*.

26 If cs is invalid, or makes an unsatisfiable request for a given input \mathbf{X} , the output shall still be an interval
27 literal whose value encloses \mathbf{X} . A language- or implementation-defined extension to interval literal syntax
28 may be used to make it obvious that this has occurred.

29 Among the user-controllable features of cs are the following.

30 a) It should be possible to specify the preferred overall field width (the length of s).

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

33 c) For l and u , it should be possible to specify the field width, and the number of digits after the point or
34 the number of significant digits.

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

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

1 7. Levels 3 and 4 description

2 This clause is about Level 3, where Level 2 datums are represented, and operations on them described, and
3 about Level 4 requirements for interchange representation and encoding.

4 Level 3 entities are called *objects*—they represent Level 2 datums and may be referred to as *concrete*, while
5 the datums are *abstract*. Each Level 2 (abstract) library operation is implemented by a corresponding Level
6 3 (concrete) operation, whose behavior shall be consistent with the abstract operation.

7 7.1 Representation

The property that defines a representation is:

Each interval datum shall be represented by at least one object. Each object shall represent at most one interval datum.

8 [Examples. Three possible representations are:

9 **inf-sup** form. Any interval x is represented at Level 3 by the object $(\text{inf}(x), \text{sup}(x))$ of two b64 numbers. All intervals
10 have only one Level 3 representation because operations `inf` and `sup` are uniquely defined at Level 2 (6.7.6): interval
11 $[0, 0]$ has representation $(-0, +0)$, interval `Empty` has representation $(+\infty, -\infty)$.

12 **inf-sup-nan** form. The objects are defined to be pairs (l, u) where l, u are b64 datums. A nonempty interval $x = [\underline{x}, \bar{x}]$
13 is represented by an object (l, u) such that the values of l and u are \underline{x} and \bar{x} , and `Empty` is represented by (NaN, NaN) .

14 **neginf-sup-nan** form. This is as the previous, except that for a nonempty interval the value of l is $-\underline{x}$.

15 If, in these descriptions l, u and `NaN` are viewed as Level 2 datums, then interval $[0, 0]$ has four representatives in **inf-sup-**
16 **nan** and **neginf-sup-nan** forms: $(-0, +0)$, $(-0, -0)$, $(+0, +0)$, $(+0, -0)$. Each nonempty interval with nonzero bounds
17 has only one representative: there are unique l and u . `Empty` has also only one representative: there is a unique `NaN`.
18 However, `NaN` itself has representatives, and from this viewpoint `Empty` has more than one representative: there are many
19 `NaN`s, quiet or signaling and with different payloads, to use in `Empty = (NaN, NaN)`.]

20 7.2 Operations and representation

21 Each Level 2 (abstract) library operation is implemented by a corresponding Level 3 (concrete) operation,
22 whose behavior shall be consistent with the abstract operation.

23 When an input Level 3 object does not represent a Level 2 datum, the result is implementation-defined. An
24 implementation shall provide means for the user to specify that an `InvalidOperand` exception be signaled
25 when this occurs.

26 7.3 Interchange representations and encodings

27 The purpose of interchange representations is to allow loss-free exchange of Level 2 interval data. This is
28 done by imposing a standard Level 3 representation using Level 2 datums.

The standard Level 3 representation of an interval datum x is an ordered pair

$$(\text{inf}(x), \text{sup}(x))$$

29 of two b64 datums. For example, the only representative of `Empty` is the pair $(+\infty, -\infty)$, and the only
30 representative of $[0, 0]$ is the pair $(-0, +0)$.

The standard Level 3 representation of a decorated interval datum x_{dx} is an ordered triple

$$(\text{inf}(x_{dx}), \text{sup}(x_{dx}), \text{decorationPart}(x_{dx}))$$

31 of two b64 datums and a decoration. For example, the only representative of `Emptytrv` is the triple
32 $(+\infty, -\infty, \text{trv})$, and the only representative of `NaN` is the triple $(\text{NaN}, \text{NaN}, \text{i11})$.

33 Export and import of interchange formats normally occurs as a sequence of **octets** (bit strings of length 8,
34 equivalently 8-bit bytes), e.g. in a file or a network packet.

1 At Level 4, interval objects are encoded as bit strings. We define an **octet-encoding** that maps the con-
2 ceptual Level 3 representation into an octet sequence that comprises, in the order defined above, the inter-
3 change octet-encodings of the two **b64** datums, and, for decorated intervals, the decoration represented as an
4 octet:

```
        ill 00000000  
        trv 00000100  
5         def 00001000  
        dac 00001100  
        com 00010000
```

6 NOTE—This encoding of decorations permits future refinements without disturbing the propagation order of the
7 decorations.

8 The octet-encoding of **b64** datums is eight octets obtained from the 64 bits of the IEEE 754 interchange
9 format: a sign bit, followed by 11 exponent bits that describe the exponent offset by a bias, and 52 bits that
10 describe the significand (the least significant bit is last).

11 In Big-Endian octet-encoding, the first octet contains the sign bit and the 7 most-significant exponent bits.

12 In Little-Endian octet-encoding, the first octet contains the 8 least-significant bits.

13 [Example. The Big-Endian interchange octet-encoding of $[-1, 3]_{\text{com}}$ are the concatenated octet sequences below

```
    -1 10111111 11110000 00000000 00000000 00000000 00000000 00000000 00000000  
14     3 01000000 00001000 00000000 00000000 00000000 00000000 00000000 00000000  
    com 00010000
```

15 The Little-Endian interchange octet-encoding of $[-1, 3]_{\text{com}}$ are the concatenated octet sequences below

```
    -1 00000000 00000000 00000000 00000000 00000000 00000000 11110000 10111111  
16     3 00000000 00000000 00000000 00000000 00000000 00000000 00001000 01000000  
    com 00010000
```

17]

ANNEX A

1 **Features of the full standard IEEE Std 1788TM-2015 not required**
2 **in the IEEE P1788.1 standard (informative)**

3 This Annex lists all features of the set-based flavor of IEEE Std 1788TM-2015 that are not required in IEEE
4 P1788.1. The corresponding subclauses in IEEE Std 1788TM-2015 are given in parentheses.

5 The following operations required in the set-based flavor of IEEE Std 1788TM-2015 are not required in IEEE
6 P1788.1:

All reverse-mode elementary functions (10.5.4)

Two-output division (10.5.5)

`mulRevToPair`

Boolean functions of intervals (10.5.10)

`less`

`precedes`

`strictLess`

`strictPrecedes`

7 *Reduction operations (12.2.12)*

`sum`

`dot`

`sumSquare`

`sumAbs`

Exact text representation (13.4)

`intervalToExact`

`exactToInterval`