IEEE STANDARDS ASSOCIATION

IEEE Standard for Interval Arithmetic (Simplified)

IEEE Computer Society

Sponsored by the
Microprocessor Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1788.1™-2017

IEEE Standard for Interval Arithmetic (Simplified)

Sponsor

Microprocessor Standards Committee
of the
IEEE Computer Society

Approved 6 December 2017

IEEE-SA Standards Board

Abstract: This standard is a simplified version and a subset of the IEEE Std 1788^{TM}-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 Std 1788.1-2017 specifies interval arithmetic operations based on intervals whose endpoints are IEEE Std 754^{TM}-2008 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 Std 1788.1-2017 should compile and run, and give identical output within round off, using an implementation of IEEE Std 1788-2015, or any superset of the former.

Compared to IEEE Std 1788-2015, 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.

Keywords: arithmetic, computing, decoration, enclosure, interval, IEEE 754^{TM}, IEEE 1788^{TM}, operation, verified.

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

PDF: ISBN 978-1-5044-4602-0 STD22946
Print: ISBN 978-1-5044-4603-7 STDPD22946

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

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

Comments on standards should be submitted to the following address:

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

Laws and regulations

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

Copyrights

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

Photocopies

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

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments,

Errata

Patents

Participants

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

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

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

Introduction

This introduction is not part of IEEE Std 1788.1$^{\text{TM}}$-2017, IEEE Standard for Interval Arithmetic (Simplified).

This standard is a simplified version of the (full) IEEE Std 1788-2015 for Interval Arithmetic. As such, IEEE Std 1788.1-2017 features

a) set-based intervals,

b) the inf-sup interval type based on the IEEE Std 754-2008 binary64 floating-point format,

c) the decoration system, and

d) simplified I/O.

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

– Finite-precision intervals of different kinds are supported, e.g. one can build a system where interval endpoints are numbers from an arbitrary-precision floating-point system such as MPFR.

– Besides the model in IEEE Std 1788.1-2017, where a mathematical interval is any closed connected subset of the real numbers, other foundational models are supported in a controlled way by the concept of *flavors*. For instance a flavor may support half-open and open real intervals; or Kaucher/modal intervals like [4,3] as well as [3,4].

  IEEE Std 1788.1-2017 satisfies all General Requirements of IEEE Std 1788-2015 and can be considered a flavor of that standard.

– The set of required operations is extended by operations useful in specialized applications, such as reverse-mode functions for constraint programming, and two-output division used in the interval Newton root-finding method.

This document consists of two parts. Part 1 contains general requirements: overview, scope and purpose of this standard and various definitions and abbreviations. Part 2 is the actual standard, presented in four levels. Level 1 summarizes the theory of set-based intervals including decorations; Level 2 is about representing Level 1 entities in finite precision and the corresponding operations; Level 3 represents Level 2 entities, and Level 4 specifies interchange encodings. Annex A lists the features of IEEE Std 1788-2015 that are not required in IEEE Std 1788.1-2017.

Contents

Part 1. General Requirements 1
1. Overview 1
1.1. Scope 1
1.2. Purpose 1
1.3. Word usage 1
1.4. Structure of the standard in levels 2
1.5. The meaning of conformance 2
2. Normative references 2
3. Special terms, abbreviations, and notation 2
3.1. Frequently used notation and abbreviations 2
3.2. Special terms 2

Part 2. Interval Standard (Simplified) 6
4. Level 1 description 6
4.1. Level 1 entities 6
4.2. Intervals 6
4.3. Hull 7
4.4. Functions 7
4.5. Required operations 8
5. The decoration system at Level 1 13
5.1. Overview 13
5.2. Definitions and properties 13
5.3. The ill-formed interval 14
5.4. Permitted combinations 14
5.5. Operations on/with decorations 14
5.6. Decorations and arithmetic operations 15
5.7. Decoration of non-arithmetic operations 15
6. Level 2 description 17
6.1. Introduction 17
6.2. Naming conventions for operations 17
6.3. Level 2 hull operation 18
6.4. Level 2 interval extensions 18
6.5. Accuracy of operations 18
6.6. Number and interval literals 19
6.7. Required operations 20
6.8. Input and output (I/O) of intervals 24
7. Levels 3 and 4 description 25
7.1. Representation 25
7.2. Operations and representation 25
7.3. Interchange representations and encodings 25

Annex A. Features of the full standard IEEE Std 1788-2015 not required in the IEEE Std 1788.1-2017 standard (informative) 27

IEEE Standard for Interval Arithmetic (Simplified)

PART 1

General Requirements

1. Overview

1.1 Scope

This standard is a simplified version and a subset of the IEEE Std 1788$^{\text{TM}}$-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 Std 1788.1-2017 specifies interval arithmetic operations based on intervals whose endpoints are IEEE Std 754$^{\text{TM}}$-2008 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 Std 1788.1-2017 should compile and run, and give identical output within round off, using an implementation of IEEE Std 1788-2015, or any superset of the former.

1.2 Purpose

Compared to IEEE Std 1788-2015, 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.]

1.4 Structure of the standard in levels

This standard is structured into four levels.

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

Level 2, *discretization* (Clause 6), approximates the mathematical theory by intervals with upper and lower bounds that are IEEE Std 754-2008 binary64 numbers and operations on such intervals. A level 2 entity is called a *datum*.[1]

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

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

1.5 The meaning of conformance

An implementation of this standard shall satisfy the following requirements:

– provide the decorations specified in 5.2;

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

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

– provide an interchange representation as specified in 7.3.

2. Normative references

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

IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic[2,3]

IEEE Std 1788-2015, IEEE Standard for Interval Arithmetic

3. Special terms, abbreviations, and notation

3.1 Frequently used notation and abbreviations

Frequently used notation and abbreviations are given in Table 3.1.

3.2 Special terms

accuracy mode A way to describe the quality of an interval version of a function. See also **tightness**.
NOTE—Details in 6.5.

[1]Plural "datums" in this standard, since "data" is often misleading.

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

[3]The 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, $\mathbb{R} \cup \{-\infty, +\infty\}$
$\overline{\mathbb{IR}}$	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
NaI	Not an Interval
NaN	Not a Number
$x, y, \ldots$ [resp. $f, g, \ldots$]	typeface/notation for a numeric value [resp. numeric function]
$\boldsymbol{x}, \boldsymbol{y}, \ldots$ [resp. $\boldsymbol{f}, \boldsymbol{g}, \ldots$]	typeface/notation for an interval value [resp. interval function]
$\mathsf{f}, \mathsf{g}, \ldots$	typeface/notation for an expression, producing a function by evaluation
$\mathrm{Dom}(f)$	the domain of a point-function f
$\mathrm{Rge}(f \mid \boldsymbol{s})$	the range of a point-function f over a set $\boldsymbol{s}$; the same as the image of $\boldsymbol{s}$ under f

arithmetic operation A **version** of a **point operation**.
NOTE—Details in 4.4.1.

bare interval Same as **interval**; used to emphasize it is not decorated.
NOTE—Details in 5.2.

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

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

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

datum One of the entities manipulated by finite precision (Level 2) operations. It can be a **boolean**, a **decoration**, a (bare) **interval**, a **decorated interval**, a **number** or a **string** datum.
NOTE—Details in 6.1.

decoration Decorations are used to record events in interval operations: e.g., evaluating a function at points where it is undefined. One of the five values `com`, `dac`, `def`, `trv` and `ill`.
NOTE—Details in Clause 5.

decorated interval A pair (interval, decoration).
NOTE—Details in Clause 5.

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

elementary function Synonymous with **arithmetic operation**.

exception An **exception** is an event that occurs, and may be signaled, when an operation on some particular operands has no outcome suitable for every reasonable application. Exceptions may occur in interval constructors and in the `intervalPart` operation.
NOTE—Details in 6.1.4, 6.7.8.

expression A symbolic form used to define a function. An **arithmetic expression** is one whose operations are all arithmetic operations.
NOTE—Details in Clause 6 of IEEE Std 1788-2015.

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

FTIA, FTDIA For a real function f defined by an arithmetic expression:

– The Fundamental Theorem of Interval Arithmetic (FTIA) gives relations between the result $\boldsymbol{y}$ of evaluating the expression in interval mode over an input box and the behavior of f on the box. The basic property is that $\boldsymbol{y}$ contains the range of f over the box; various extra conditions can give extra conclusions, such as that f is continuous on the box.

– The Fundamental Theorem of Decorated Interval Arithmetic (FTDIA) describes how evaluating the expression in decorated interval mode enables the various conditions of the FTIA to be verified, making the corresponding conclusions computable.

NOTE—Details in 6.4 of IEEE Std 1788-2015.

function Has the usual mathematical meaning of a (possibly partial, i.e., not everywhere defined) function. Synonymous with **map**, **mapping**.

hull (or **interval hull**) The hull of a subset $\boldsymbol{s}$ of $\mathbb{R}$ is the tightest interval containing $\boldsymbol{s}$.

implementation When used without qualification, means a realization of an interval arithmetic conforming to the specification of this standard.

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

interval At Level 1, a (bare) interval $\boldsymbol{x}$ is a closed connected subset of $\mathbb{R}$. At Level 2 a $\mathbb{T}$-interval is a member of the bare interval type $\mathbb{T}$, or a non-NaI member of the decorated interval type $\mathbb{T}$.
NOTE—Details in 4.2, 6.

interval extension At Level 1, an interval extension of a point function f is a function $\boldsymbol{f}$ from intervals to intervals such that $f(x)$ belongs to $\boldsymbol{f}(\boldsymbol{x})$ whenever x belongs to $\boldsymbol{x}$ and $f(x)$ is defined. It is the **natural** (or tightest) interval extension, if $\boldsymbol{f}(\boldsymbol{x})$ is the interval hull of the range of f over $\boldsymbol{x}$, for all $\boldsymbol{x}$.
NOTE—Details in 4.4.4, for Level 2 interval extension, see 6.4.

decorated interval extension of f is a function from decorated intervals to decorated intervals, whose interval part is an interval extension of f, and whose decoration part propagates decorations as specified in 5.6.
NOTE—Details in Clause 5.

interval vector See **box**.

library The set of Level 1 operations (Level 1 library) or those provided by an implementation (Level 2 library). Further classification may be made into the **point library**, **bare interval library** and **decorated interval library**.

map, mapping See **function**.

mathematical interval of constructor The arguments of an interval constructor, if valid, define a mathematical interval $\boldsymbol{x}$. The actual interval returned by the constructor is the tightest interval that contains $\boldsymbol{x}$.
NOTE—Details in 6.7.5.

NaI, NaN At Level 1 NaN is the function $\mathbb{R}^0 \to \mathbb{R}$ with empty domain. NaI is the natural interval extension 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 the Not a Number datum. NaI is the Not an Interval datum.
NOTE—Details in 5.3.

no value A mathematical (Level 1) operation evaluated at a point outside its domain is said to have no value. Used instead of "undefined", which can be ambiguous. E.g., in this standard, real number division x/y has no value when $y = 0$.
NOTE—Details in 6.1.4.

non-arithmetic operation An operation on intervals that is not an interval version of a point function; includes intersection and convex hull of two intervals.
NOTE—Details in 5.7.

number Any member of the set $\mathbb{R} \cup \{-\infty, +\infty\}$ of extended reals: a **finite number** if it belongs to $\mathbb{R}$, else an **infinite number**.

octet Bit string of length 8, equivalently 8-bit byte.

An **octet-encoding** is a mapping from the conceptual bit string encodings of floating-point datums and of decorations into an octet sequence.
NOTE—Details in 7.3.

point function A mathematical (Level 1) function of real variables.
NOTE—Details in 4.4.1.

range The range, $\mathrm{Rge}(f \mid \boldsymbol{s})$, of a function f over a set $\boldsymbol{s}$ is the set of values $f(x)$ at those points of $\boldsymbol{s}$ where f is defined.
NOTE—Details in 4.4.2.

string, text A **text string**, or just **string**, is a finite character sequence belonging to some alphabet. The term **text** is also used to mean strings generally.

tightest Smallest in the partial order of set containment. The tightest set (unique, if it exists) with a given property is contained in every other set with that property.
Also denotes one of the accuracy modes, see **tightness**.

tightness The strongest accuracy mode (these are *tightest*, *accurate*, *valid*) that a given operation achieves for some input box, or uniformly over some set of inputs.
NOTE—Details in 6.5.1.

type (or **interval type**) One of the sets into which bare and decorated interval datums are organized at Level 2, usually regarded as a finite set $\mathbb{T}$ of Level 1 intervals, (a **bare type**) in the bare case; and of Level 1 decorated intervals together with the value NaI (a **decorated type**).

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 $\mathbb{T}$-datum, and for decorated types means a non-NaI member of $\mathbb{T}$.
NOTE—Details in 6.1.

PART 2

Interval Standard (Simplified)

4. Level 1 description

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

4.1 Level 1 entities

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

– The set $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ of **extended reals**.

– The set of **(text) strings**, namely finite sequences of **characters** chosen from some alphabet.

– The set of **integers** $\mathbb{Z}$.

– The **boolean** values `false` and `true`.

– The set of **decorations** (defined in 5).

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**. An interval's members are finite numbers, but its bounds can be infinite. Finite or infinite numbers can be inputs to interval constructors, as well as outputs from operations, e.g., the interval width operation.

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

4.2 Intervals

The set of mathematical intervals is denoted by $\overline{\mathbb{IR}}$. It consists of exactly those subsets $\boldsymbol{x}$ 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

$$[\underline{x}, \overline{x}] = \{\, x \in \mathbb{R} \mid \underline{x} \le x \le \overline{x} \,\}, \tag{1}$$

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

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

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

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

4.3 Hull

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

4.4 Functions

4.4.1 Function terminology

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

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

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

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

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

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

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

4.4.2 Point function

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

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

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

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

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

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

Equivalently, for the case where f takes separate arguments $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$, each being a subset of $\mathbb{R}$, the range is written as $\operatorname{Rge}(f \mid \boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$.

4.4.3 Point arithmetic operation

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

4.4.4 Interval-valued functions

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

$$\boldsymbol{f}(\boldsymbol{x}) \supseteq \operatorname{Rge}(f \mid \boldsymbol{x})$$

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

The **tightest interval extension** of f is the mapping $\boldsymbol{f}$ defined by

$$\boldsymbol{f}(\boldsymbol{x}) = \operatorname{hull}\bigl(\operatorname{Rge}(f \mid \boldsymbol{x})\bigr).$$

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

$$\boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \supseteq \operatorname{Rge}(f \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n),$$

and the tightest interval extension is defined by

$$\boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \operatorname{hull}\bigl(\operatorname{Rge}(f \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)\bigr)$$

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

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

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

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

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

An interval-valued function in the library is called an

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

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

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

4.4.5 Constants

A real scalar function with no arguments—a mapping $\mathbb{R}^n \to \mathbb{R}^m$ with $n = 0$ and $m = 1$—is a **real constant**. An interval extension of a real constant is any zero-argument interval function that returns an interval containing c. The *tightest extension* returns the interval $[c, c]$.

4.5 Required operations

4.5.1 Interval constants

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

4.5.2 Arithmetic operations

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

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

Table 4.1. Required forward elementary functions.

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

4.5.3 Cancellative addition and subtraction

For intervals $\boldsymbol{x}$ and $\boldsymbol{y}$, cancellative subtraction, `cancelMinus`$(\boldsymbol{x}, \boldsymbol{y})$, determines the tightest interval $\boldsymbol{z}$ such that

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

if such a $\boldsymbol{z}$ exists.

At Level 1,

$$\boldsymbol{z} = \begin{cases} \emptyset & \text{if } \boldsymbol{x} = \emptyset \text{ and } \boldsymbol{y} \text{ is bounded} \\ [\underline{x} - \underline{y},\ \overline{x} - \overline{y}] & \text{if } \boldsymbol{x} \text{ and } \boldsymbol{y} \text{ are both nonempty and bounded and } \overline{y} - \underline{y} \leq \overline{x} - \underline{x}, \end{cases} \tag{2}$$

and `cancelMinus`$(\boldsymbol{x}, \boldsymbol{y})$ has no value in the cases when

$$\begin{cases} \text{either of } \boldsymbol{x} \text{ or } \boldsymbol{y} \text{ is unbounded, or} \\ \boldsymbol{x} \neq \emptyset \text{ and } y = \emptyset\text{, or,} \\ \boldsymbol{x} \text{ and } \boldsymbol{y} \text{ are both nonempty and bounded and } \overline{y} - \underline{y} > \overline{x} - \underline{x}. \end{cases} \tag{3}$$

The operation `cancelPlus`$(\boldsymbol{x}, \boldsymbol{y})$ is equivalent to `cancelMinus`$(\boldsymbol{x}, -\boldsymbol{y})$, and therefore not considered separately.

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
`intersection`$(\boldsymbol{a}, \boldsymbol{b})$	intersection $\boldsymbol{a} \cap \boldsymbol{b}$ of the intervals $\boldsymbol{a}$ and $\boldsymbol{b}$
`convexHull`$(\boldsymbol{a}, \boldsymbol{b})$	interval hull of the union $\boldsymbol{a} \cup \boldsymbol{b}$ of the intervals $\boldsymbol{a}$ and $\boldsymbol{b}$

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:

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

where l and u are extended-real values; and

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

4.5.6 Numeric functions of intervals

The operations in Table 4.3 shall be provided, the argument being an interval and the result a number, which for some of the operations may be infinite.

Implementations should provide an operation that returns `mid`$(\boldsymbol{x})$ and `rad`$(\boldsymbol{x})$ simultaneously.

Table 4.3. Required numeric functions of an interval $\boldsymbol{x} = [\underline{x}, \overline{x}]$.
Note: `inf` can have value $-\infty$; each of `sup`, `wid`, `rad` and `mag` can have value $+\infty$.

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

4.5.7 Boolean functions of intervals

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

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

Name	Returns
`isEmpty`$(\boldsymbol{x})$	1 if $\boldsymbol{x}$ is the empty set, 0 otherwise
`isEntire`$(\boldsymbol{x})$	1 if $\boldsymbol{x}$ is the whole line, 0 otherwise

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

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

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

Table 4.6. Comparisons with empty intervals.

	$\boldsymbol{a} = \emptyset$, $\boldsymbol{b} \neq \emptyset$	$\boldsymbol{a} \neq \emptyset$, $\boldsymbol{b} = \emptyset$	$\boldsymbol{a} = \emptyset$, $\boldsymbol{b} = \emptyset$
$\boldsymbol{a} = \boldsymbol{b}$	0	0	1
$\boldsymbol{a} \subseteq \boldsymbol{b}$	1	0	1
$\boldsymbol{a} \Subset \boldsymbol{b}$	1	0	1
$\boldsymbol{a} \not\cap \boldsymbol{b}$	1	1	1

5. The decoration system at Level 1

5.1 Overview

An implementation makes the decoration system available by providing:

– a decorated version of each interval extension of an arithmetic operation, of each interval constructor, and of some other operations; and

– various auxiliary functions, e.g., to extract the interval and decoration parts, and to apply a standard initial decoration to an interval.

The decoration system is specified here at a mathematical level, with the finite-precision aspects presented 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 an initial decoration, and how decorations are bound to library interval arithmetic operations to give correct propagation through expressions; 5.7 is about non-arithmetic operations.

5.2 Definitions and properties

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

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

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

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

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

(5)

These are listed according to the propagation order (10), which may also be thought of as a quality-order of $(f, \boldsymbol{x})$ pairs—decorations above `trv` are "good" and `ill` is "bad".

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

The set of decorated intervals is denoted by $\overline{\mathbb{DIR}}$, and the set of decorated boxes with n components is denoted by $\overline{\mathbb{DIR}}^n$.

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

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 $(f, \boldsymbol{x})$ pairs, and the others are nonempty subsets of `trv`. By design, they satisfy the **exclusivity rule**

For any two decorations, either one contains the other or they are disjoint. (6)

Namely, the definitions (5) give:

$$\texttt{com} \subset \texttt{dac} \subset \texttt{def} \subset \texttt{trv} \supset \texttt{ill}, \qquad \text{note the change from } \subset \text{ to } \supset \tag{7}$$

$$\texttt{com}, \texttt{dac} \text{ and } \texttt{def} \text{ are disjoint from } \texttt{ill}. \tag{8}$$

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

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

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

5.3 The ill-formed interval

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

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

– a constructor when it cannot construct a valid decorated interval, or of

– a library arithmetic operation if and only if one of its inputs is ill-formed.

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

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

Information may be stored in a NaI in an implementation-defined way (like the payload of an IEEE Std 754-2008 floating-point NaN), and functions may be provided for a user to set and read this for diagnostic purposes. An implementation may provide means for an exception to be signaled when a NaI is produced.

5.4 Permitted combinations

A decorated interval $\boldsymbol{y}_{dy}$ shall always be such that

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

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

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

are contradictory—implementations shall not produce them.

No other combinations are forbidden.

5.5 Operations on/with decorations

5.5.1 Initializing

A bare interval is initialized with a decoration by the operation

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

5.5.2 Disassembling and assembling

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

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

– $\texttt{setDec}(\emptyset, dx)$, where dx is one of $\texttt{def}$, $\texttt{dac}$ or $\texttt{com}$, returns $\emptyset_{\texttt{trv}}$;

– $\texttt{setDec}(\boldsymbol{x}, \texttt{com})$, for any unbounded $\boldsymbol{x}$, returns $\boldsymbol{x}_{\texttt{dac}}$; and

– $\texttt{setDec}(\boldsymbol{x}, \texttt{ill})$ for any $\boldsymbol{x}$, whether empty or not, returns NaI.

5.5.3 Comparisons

For decorations, comparison operations for equality $=$ and its negation $\neq$ shall be provided, as well as comparisons $>, <, \geq, \leq$ with respect to the propagation order (10) shown below.

5.6 Decorations and arithmetic operations

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

The definition of a bare interval extension implies

$$\boldsymbol{w} \supseteq \mathrm{Rge}(\varphi \,|\, \boldsymbol{v}) \qquad \text{(enclosure).}$$

The decorated interval extension of φ determines a dv_0 such that

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

It then evaluates the output decoration dw by

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

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

$$\texttt{com} > \texttt{dac} > \texttt{def} > \texttt{trv} > \texttt{ill}. \tag{10}$$

5.7 Decoration of non-arithmetic operations

5.7.1 Interval-valued operations

These give interval results but are not interval extensions of point functions:

– the cancellative operations $\texttt{cancelPlus}(\boldsymbol{x}, \boldsymbol{y})$ and $\texttt{cancelMinus}(\boldsymbol{x}, \boldsymbol{y})$ of 4.5.3;

– the set-oriented operations $\texttt{intersection}(\boldsymbol{x}, \boldsymbol{y})$ and $\texttt{convexHull}(\boldsymbol{x}, \boldsymbol{y})$ of 4.5.4

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

5.7.2 Non-interval-valued operations

These give non-interval results:

– the numeric functions of 4.5.6 and

– the boolean-valued functions of 4.5.7.

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

6. Level 2 description

6.1 Introduction

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

6.1.1 Interval type

The interval type, denoted by $\mathbb{T}$, is the **inf-sup type** derived from the IEEE Std 754-2008 `binary64` format; we refer to the latter as `b64`. This interval type comprises all intervals whose endpoints are `b64` numbers, together with Empty. Since $\pm\infty$ are in `b64`, Entire is in $\mathbb{T}$.

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 to an entity that can be a bare or decorated interval, including NaI. A **$\mathbb{T}$-box** is a vector with $\mathbb{T}$-datum components.

6.1.2 Decorated interval type

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 this type by $\mathbb{DT}$.

$\mathbb{DT}$ shall contain a "Not an Interval" datum NaI, which is identified with $(\emptyset, \texttt{ill})$.

6.1.3 Operations

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

– An interval extension (see 6.4) of one of the arithmetic operations of 4.5.

– A set operation, such as intersection and convex hull of $\mathbb{T}$-intervals, returning a $\mathbb{T}$-interval.

– A function such as the midpoint, whose input is a $\mathbb{T}$-interval and output is numeric.

– A constructor, whose input is numeric or text and output is a $\mathbb{T}$-datum.

6.1.4 Exception behavior

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

– a special value indicating this event (e.g., NaN for most of the numeric functions in 6.7.6); or

– a value considered reasonable in practice. For example, `mid`(Entire) returns 0; a constructor given invalid input returns Empty; and one of the comparisons of 6.7.7, if any input is NaI, returns `false`.

If `intervalPart`() is called with NaI as input, the exception `IntvlPartOfNaI` is signaled (see 6.7.8).

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

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

6.2 Naming conventions for operations

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

[4]Not "data", whose common meaning could cause confusion.

6.3 Level 2 hull operation

6.3.1 Hull in one dimension

The **interval hull** operation

$$\boldsymbol{y} = \operatorname{hull}(\boldsymbol{s}),$$

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

6.3.2 Hull in n dimensions

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

$$\boldsymbol{y}_i = \operatorname{hull}(\boldsymbol{s}_i),$$

and $\boldsymbol{s}_i = \{\, s_i \mid s \in \boldsymbol{s} \,\}$ is the projection of $\boldsymbol{s}$ on the ith coordinate dimension.

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 $\boldsymbol{f}$ from n-dimensional $\mathbb{T}$-boxes to $\mathbb{T}$-intervals, that is $\boldsymbol{f} : \mathbb{T}^n \to \mathbb{T}$, such that $f(x) \in \boldsymbol{f}(\boldsymbol{x})$ whenever $x \in \boldsymbol{x}$ and $f(x)$ is defined. Equivalently

$$\boldsymbol{f}(\boldsymbol{x}) \supseteq \operatorname{Rge}(f \,|\, \boldsymbol{x}),$$

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

6.5 Accuracy of operations

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

6.5.1 Measures of accuracy

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

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

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

$$\boldsymbol{f}(\boldsymbol{x}) \supseteq \boldsymbol{f}_{\text{exact}}(\boldsymbol{x}). \tag{11}$$

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

$$\boldsymbol{f}_{\text{tightest}}(\boldsymbol{x}) = \operatorname{hull}\bigl(\boldsymbol{f}_{\text{exact}}(\boldsymbol{x})\bigr). \tag{12}$$

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

$$\boldsymbol{f}(\boldsymbol{x}) \subseteq \texttt{nextOut}\left(\boldsymbol{f}_{\text{tightest}}\left(\texttt{nextOut}\bigl(\operatorname{hull}(\boldsymbol{x})\bigr)\right)\right). \tag{13}$$

For an interval $\boldsymbol{x}$,

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

where `nextUp` and `nextDown` are equivalent to the corresponding functions in IEEE Std 754-2008.

When $\boldsymbol{x}$ is an interval box, `nextOut` acts componentwise.

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

6.5.2 Accuracy requirements

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

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

6.5.3 Documentation requirements

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

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

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

]

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

6.6 Number and interval literals

6.6.1 Number literals

The following forms of number literal shall be provided.

a) A decimal number. This comprises an optional sign, a nonempty sequence of decimal digits optionally containing a point, and an optional exponent field comprising `e` and an integer literal.[5] The value of a decimal number is the value of the sequence of decimal digits with optional point multiplied by ten raised to the power of the value of the integer literal, negated if there is a leading `-` sign.

b) A number in the hexadecimal-floating-constant form of the C99 standard (ISO/IEC9899, N1256 (6.4.4.2)), equivalently hexadecimal-significand form of IEEE Std 754-2008 (5.12.3). This comprises an optional sign, the string `0x`, a nonempty sequence of hexadecimal digits optionally containing a point, and an exponent field comprising `p` and an integer literal exponent. The value of a hexadecimal number is the value of the sequence of hexadecimal digits with optional point multiplied by two raised to the power of the value of the exponent, negated if there is a leading minus sign.

c) A rational literal $p \, / \, q$. This comprises an integer literal p, the `/` character, and a strictly positive integer literal q. Its value is the value of p divided by the value of q.

d) Either of the strings `inf` or `infinity` optionally preceded by `+`, with value $+\infty$; or preceded by `-`, with value $-\infty$.

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

6.6.2 Bare intervals

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

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

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

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

– Special values: the strings `[ ]` and `[ empty ]`, whose bare value is Empty, and the string `[ entire ]`, whose bare value is Entire.

6.6.3 Decorated intervals

The following forms of decorated interval literal shall be supported.

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

  If ***sx*** has the bare value $\boldsymbol{x}$, and if $\boldsymbol{x}_{dx}$ is a permitted combination according to 5.4, then ***sx*_*sd*** has the value $\boldsymbol{x}_{dx}$. Otherwise ***sx*_*sd*** has no value as a decorated interval literal.

– The string `[ nai ]`, with the bare value Empty and the decorated value $\text{Empty}_{\texttt{ill}}$.

The alphanumeric characters in the above literals are case-insensitive (e.g., `[1,1e3]_com` is equivalent to `[1,1E3]_COM`).

6.7 Required operations

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

6.7.1 Interval constants

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

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

respectively.

6.7.2 Elementary functions

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

NOTE—For operations, some of whose arguments are of integer type, such as integer power $\texttt{pown}(x, p)$, only the real arguments are replaced by intervals.

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

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

6.7.3 Cancellative addition and subtraction

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

`cancelMinus`$(\boldsymbol{x}, \boldsymbol{y})$ shall return Empty in the first case of (2), the hull of the result in the second, and Entire for each of the cases in (3).

`cancelPlus`$(\boldsymbol{x}, \boldsymbol{y})$ shall be equivalent to `cancelMinus`$(\boldsymbol{x}, -\boldsymbol{y})$.

These operations shall have "trivial" decorated versions, as described in 5.7.

6.7.4 Set operations

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

These operations shall have "trivial" decorated versions, as described in 5.7.

6.7.5 Constructors

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

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

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

For the constructor `textToInterval`$(\boldsymbol{s})$, the input $\boldsymbol{s}$ is a string. The string of form `[` l `,` u `]` where $l < +\infty$ and $u > -\infty$ are optional number literals is called **accuracy-relaxed** if either one of the literals is a rational 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 is a hexadecimal number literal of form b) in 6.6.1.

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

– If $\boldsymbol{s}$ is not a valid interval interval and $\boldsymbol{s}$ is not accuracy-relaxed, this constructor fails, and the result is Empty.

– If $\boldsymbol{s}$ is accuracy-relaxed and $l \le u$, the result may be any interval containing `[`l`,`u`]` (the constructor succeeds).

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

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

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

Otherwise the call fails, and the result is NaI.

Exception behavior. Exception `UndefinedOperation` is signaled by both the bare and the decorated constructor when the input is such that the bare constructor fails.

Exception `PossiblyUndefinedOperation` is signaled by both the bare and the decorated `textToInterval`($\boldsymbol{s}$) constructor with accuracy-relaxed $\boldsymbol{s}$

– when $l > u$ and interval constructor succeeds;

– when $l \leq u$ and the result is wider than the hull of `[`l`,`u`]`;

– optionally when $l \leq u$ and the result is the hull of `[`l`,`u`]`.

NOTE—When signaled by the decorated constructor it will normally be ignored since returning NaI gives sufficient information.

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

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

6.7.6 Numeric functions of intervals

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

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

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

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

NOTE—These functions help define operations of the standard but are not themselves operations of the standard.

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

`inf`($\boldsymbol{x}$) returns the Level 1 value rounded toward negative.

`sup`($\boldsymbol{x}$) returns the Level 1 value rounded toward positive.

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

$\boldsymbol{x} =$ Empty	NaN
$\boldsymbol{x} =$ Entire	0
$\underline{x} = -\infty$, $\overline{x}$ finite	the finite negative `b64` number of largest magnitude
$\underline{x}$ finite, $\overline{x} = +\infty$	the finite positive `b64` number of largest magnitude
$\underline{x}$, $\overline{x}$ both finite	the Level 1 value rounded to nearest

The implementation shall document how it handles the last case.

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

`wid`($\boldsymbol{x}$) returns NaN if $\boldsymbol{x}$ is empty. Otherwise it returns the Level 1 value rounded toward positive.

`mag`($\boldsymbol{x}$) returns NaN if $\boldsymbol{x}$ is empty. Otherwise it returns the Level 1 value rounded toward positive.

`mig`($\boldsymbol{x}$) returns NaN if $\boldsymbol{x}$ is empty. Otherwise it returns the Level 1 value rounded toward negative, except that 0 maps to +0.

Each bare interval operation in this subclause shall have a decorated version, where each input of bare interval type is replaced by one of the corresponding decorated interval type, and the result format is that of the bare operation. Following 5.7, if any input is NaI, the result is NaN. Otherwise the result is obtained by discarding the decoration and applying the corresponding bare interval operation.

6.7.7 Boolean functions of intervals

An implementation shall provide the functions

– `isEmpty`($\boldsymbol{x}$) and `isEntire`($\boldsymbol{x}$) in 4.5.7,

– the functions implementing the comparison relations in Table 4.5 of 4.5.7, and

– the function `isNaI`($\boldsymbol{x}$) for input $\boldsymbol{x}$ of the decorated type, which returns `true` if $\boldsymbol{x}$ is NaI, else `false`.

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

6.7.8 Operations on/with decorations

An implementation shall provide the operations of 5.5. These comprise the comparison operations $=$, $\neq$, $>$, $<$, $\geq$, $\leq$ for decorations; and, for the decorated type, the operations `newDec`, `intervalPart`, `decorationPart` and `setDec`.

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

6.8 Input and output (I/O) of intervals

6.8.1 Overview

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

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

6.8.2 Input

Input is provided by `textToInterval`($\boldsymbol{s}$); see 6.7.5.

6.8.3 Output

An implementation shall provide an operation

$$\texttt{intervalToText}(\boldsymbol{X}, cs)$$

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

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

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

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

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

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

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

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

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

a) It should be possible to specify the preferred overall field width (the length of $\boldsymbol{s}$).

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

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

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

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

7. Levels 3 and 4 description

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

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

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.

[Examples. Three possible representations are:

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

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

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

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

7.2 Operations and representation

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

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

7.3 Interchange representations and encodings

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

The standard Level 3 representation of an interval datum $\boldsymbol{x}$ is an ordered pair

$$\big(\texttt{inf}(x),\ \texttt{sup}(x)\big)$$

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

The standard Level 3 representation of a decorated interval datum $\boldsymbol{x}_{dx}$ is an ordered triple

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

of two b64 datums and a decoration. For example, the only representative of $\mathrm{Empty}_{\texttt{trv}}$ is the triple $(+\infty,-\infty,\texttt{trv})$, and the only representative of NaI is the triple $(\mathrm{NaN},\mathrm{NaN},\texttt{ill})$.

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

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

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

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

The octet-encoding of `b64` datums is eight octets obtained from the 64 bits of the IEEE Std 754-2008 interchange format: a sign bit, followed by 11 exponent bits that describe the exponent offset by a bias, and 52 bits that describe the significand (the least significant bit is last).

In Big-Endian octet-encoding, the first octet contains the sign bit and the 7 most-significant exponent bits. In Little-Endian octet-encoding, the first octet contains the 8 least-significant bits.

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

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

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

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

]

ANNEX A

Features of the full standard IEEE Std 1788-2015 not required in the IEEE Std 1788.1-2017 standard (informative)

This Annex lists all features of the set-based flavor of IEEE Std 1788-2015 that are not required in IEEE Std 1788.1-2017. The corresponding subclauses in IEEE Std 1788-2015 are given in parentheses.

The following operations required in the set-based flavor of IEEE Std 1788-2015 are not required in IEEE Std 1788.1-2017:

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`
Reduction operations (12.2.12)
`sum`
`dot`
`sumSquare`
`sumAbs`
Exact text representation (13.4)
`intervalToExact`
`exactToInterval`

Consensus

WE BUILD IT.

IEEE
standards.ieee.org
Phone: +1 732 981 0060   Fax: +1 732 562 1571
© IEEE