

BS ISO/IEC 10967-1:2012



BSI Standards Publication

Information technology — Language independent arithmetic

Part 1: Integer and floating point arithmetic

bsi.

...making excellence a habit.
AVA Documents™
Document Sharing Hub

National foreword

This British Standard is the UK implementation of ISO/IEC 10967-1:2012. It supersedes BS ISO/IEC 10967-1:1994 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee IST/5, Programming languages, their environments and system software interfaces.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2012. Published by BSI Standards Limited 2012

ISBN 978 0 580 63696 7

ICS 35.060

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 July 2012.

Amendments issued since publication

Date	Text affected
------	---------------

**Information technology — Language
independent arithmetic —**

**Part 1:
Integer and floating point arithmetic**

*Technologies de l'information — Arithmétique indépendante de
langage —*

Partie 1: Arithmétique de nombres entiers et en virgule flottante



Reference number
ISO/IEC 10967-1:2012(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2012



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2012

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Foreword	vii
Introduction	viii
1 Scope	1
1.1 Inclusions	1
1.2 Exclusions	2
2 Conformity	3
3 Normative references	4
4 Symbols and definitions	4
4.1 Symbols	4
4.1.1 Operators and relations	4
4.1.2 Sets and intervals	5
4.1.3 Exceptional values	5
4.1.4 Special values	6
4.1.5 The Boolean datatype	6
4.1.6 Operation specification framework	6
4.2 Definitions of terms	7
5 Specifications for integer and floating point datatypes and operations	12
5.1 Integer datatypes and operations	13
5.1.1 Integer result function	14
5.1.2 Integer operations	14
5.1.2.1 Comparisons	14
5.1.2.2 Basic arithmetic	15
5.2 Floating point datatypes and operations	17
5.2.1 Conformity to IEC 60559	19
5.2.2 Range and granularity constants	19
5.2.3 Approximate operations	19
5.2.4 Rounding and rounding constants	20
5.2.5 Floating point result function	21
5.2.6 Floating point operations	22
5.2.6.1 Comparisons	22
5.2.6.2 Basic arithmetic	24
5.2.6.3 Value dissection	27
5.2.6.4 Value splitting	29
5.3 Operations for conversion between numeric datatypes	29
5.3.1 Integer to integer conversions	30
5.3.2 Floating point to integer conversions	31
5.3.3 Integer to floating point conversions	31
5.3.4 Floating point to floating point conversions	32
5.3.5 Floating point to fixed point conversions	32
5.3.6 Fixed point to floating point conversions	34
5.4 Numerals as operations in a programming language	34

5.4.1	Numerals for integer datatypes	34
5.4.2	Numerals for floating point datatypes	35
6	Notification	35
6.1	Model for handling of notifications	35
6.2	Notification alternatives	36
6.2.1	Notification by recording in indicators	36
6.2.2	Notification by alteration of control flow	38
6.2.3	Notification by termination with message	38
6.3	Delays in notification	39
6.4	User selection of alternative for notification	39
7	Relationship with language standards	39
8	Documentation requirements	41
Annex A	(informative) Partial conformity	43
A.1	Integer overflow notification relaxation	44
A.2	Infinitary notification relaxation	44
A.3	Inexact notification relaxation	44
A.4	Underflow notification relaxation	45
A.5	Subnormal values relaxation	45
A.6	Accuracy relaxation for add, subtract, multiply, and divide	45
A.7	Accuracy relaxation for floating point conversion	47
Annex B	(informative) IEC 60559 bindings	51
B.1	Summary	51
B.2	Notification	55
Annex C	(informative) Rationale	57
C.1	Scope	57
C.1.1	Inclusions	57
C.1.2	Exclusions	57
C.1.3	Companion parts to this part	58
C.2	Conformity	58
C.2.1	Validation	59
C.3	Normative references	59
C.4	Symbols and definitions	59
C.4.1	Symbols	60
C.4.2	Definitions of terms	60
C.5	Specifications for integer and floating point datatypes and operations	61
C.5.1	Integer datatypes and operations	62
C.5.1.0.1	Unbounded integers	62
C.5.1.0.2	Bounded non-modulo integers	63
C.5.1.0.3	Modulo integers	64
C.5.1.1	Integer result function	64
C.5.1.2	Integer operations	64
C.5.1.2.1	Comparisons	64
C.5.1.2.2	Basic arithmetic	65

C.5.2	Floating point datatypes and operations	65
C.5.2.0.1	Constraints on the floating point parameters	66
ISO/IEC 10967-1:2012(E)	C.5.2.0.2 Radix complement floating point	68
C.5.2.1	Conformity to IEC 60559	68
C.5.2.1.1	Subnormal numbers	69
C.5.2.1.2	Signed zero	69
C.5.2.1.3	Infinities and NaNs	69
C.5.2.2	Range and granularity constants	70
C.5.2.2.1	Relations among floating point datatypes	70
C.5.2.3	Approximate operations	71
C.5.2.4	Rounding and rounding constants	71
C.5.2.5	Floating point result function	73
C.5.2.6	Floating point operations	73
C.5.2.6.1	Comparisons	73
C.5.2.6.2	Basic arithmetic	73
C.5.2.6.3	Value dissection	74
C.5.2.6.4	Value splitting	74
C.5.2.7	Levels of predictability	75
C.5.2.8	Identities	76
C.5.2.9	Precision, accuracy, and error	78
C.5.2.9.1	LIA-1 and error	79
C.5.2.9.2	Empirical and modelling errors	80
C.5.2.9.3	Propagation of errors	80
C.5.2.10	Extra precision	81
C.5.3	Operations for conversion between numeric datatypes	82
C.5.4	Numerals as operations in a programming language	83
C.5.4.1	Numerals for integer datatypes	83
C.5.4.2	Numerals for floating point datatypes	83
C.6	Notification	83
C.6.1	Model handling of notifications	84
C.6.2	Notification alternatives	84
C.6.2.1	Notification by recording in indicators	84
C.6.2.2	Notification by alteration of control flow	85
C.6.2.3	Notification by termination with message	86
C.6.3	Delays in notification	86
C.6.4	User selection of alternative for notification	86
C.7	Relationship with language standards	87
C.8	Documentation requirements	88
Annex D	(informative) Example bindings for specific languages	89
D.1	Ada	90
D.2	C	96
D.3	C++	104
D.4	Fortran	111
D.5	Common Lisp	115
Annex E	(informative) Example of a conformity statement	121
E.1	Types	121

E.2	Integer parameters	121
E.3	Floating point parameters	122
E.4	Expressions	122
E.5	Notification	122
Annex F	(informative) Example programs	125
F.1	Verifying platform acceptability	125
F.2	Selecting alternate code	125
F.3	Terminating a loop	126
F.4	Estimating error	126
F.5	Saving exception state	126
F.6	Fast versus accurate	127
F.7	High-precision multiply	127
Bibliography		129

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 10967-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

This second edition cancels and replaces the first edition (ISO/IEC 10967-1:1994), which has been technically revised.

ISO/IEC 10967-1 consists of the following parts, under the general title *Information technology — Language independent arithmetic*:

- *Part 1: Integer and floating point arithmetic*
- *Part 2: Elementary numerical functions*
- *Part 3: Complex integer and floating point arithmetic and complex elementary numerical functions*

Introduction

The aims

Programmers writing programs that perform a significant amount of numeric processing have often not been certain how a program will perform when run under a given language processor. Programming language standards have traditionally been somewhat weak in the area of numeric processing, seldom providing an adequate specification of the properties of arithmetic datatypes, particularly floating point numbers. Often they do not even require much in the way of documentation of the actual arithmetic datatypes by a conforming language processor.

It is the intent of this part of ISO/IEC 10967 to help to redress these shortcomings, by setting out precise definitions of integer and floating point datatypes, and requirements for documentation.

It is not claimed that this part of ISO/IEC 10967 will ensure complete certainty of arithmetic behaviour in all circumstances; the complexity of numeric software and the difficulties of analysing and proving algorithms are too great for that to be attempted.

The first aim of this part of ISO/IEC 10967 is to enhance the predictability and reliability of the behaviour of programs performing numeric processing.

The second aim, which helps to support the first, is to help programming language standards to express the semantics of arithmetic datatypes.

The third aim is to help enhance the portability of programs that perform numeric processing across a range of different platforms. Improved predictability of behaviour will aid programmers designing code intended to run on multiple platforms, and will help in predicting what will happen when such a program is moved from one conforming language processor to another.

Note that this part of ISO/IEC 10967 does not attempt to ensure bit-for-bit identical results when programs are transferred between language processors, or translated from one language into another. However, experience shows that diverse numeric environments can yield comparable results under most circumstances, and that with careful program design significant portability is actually achievable. In addition, the IEC 60559 (IEEE 754) standard goes a long way to ensure bit-for-bit identical results, and in this second edition of this part of ISO/IEC 10967 the requirements are tightened (compared to the first edition) to approach those of IEEE 754.

The content

This part of ISO/IEC 10967 defines the fundamental properties of integer and floating point datatypes. These properties are presented in terms of a parameterised model. The parameters allow enough variation in the model so that several integer and floating point datatypes are covered. In particular, the IEC 60559 (IEEE 754) floating point datatypes, both those of radix 2 and those of radix 10, are covered, as well as integer datatypes, both unlimited and limited, for the latter both signed or unsigned, are covered. But when a particular set of parameter values is selected, and all required documentation is supplied, the resulting information should be precise enough to permit careful numerical analysis.

The requirements of this part of ISO/IEC 10967 cover four areas. First, the programmer must be given runtime access to the specified operations on values of integer or floating point datatype. Second, the programmer must be given runtime access to the parameters (and parameter functions) that describe the arithmetic properties of an integer or floating point datatype. Third, the executing program must be notified when proper results cannot be returned (e.g., when a

computed result may be out of range or undefined). Fourth, the numeric properties of conforming platforms must be publicly documented.

This part of ISO/IEC 10967 focuses on the classical integer and floating point datatypes. Subsequent parts considers common elementary numerical functions (Part 2), complex numerical numbers and complex elementary numerical functions (Part 3).

The benefits

Adoption and proper use of this part of ISO/IEC 10967 can lead to the following benefits.

For programming language standards it will be possible to define their arithmetic semantics more precisely without preventing the efficient implementation of the language on a wide range of machine architectures.

Programmers of numeric software will be able to assess the portability of their programs in advance. Programmers will be able to trade off program design requirements for portability in the resulting program.

In programs one will be able to determine (at run time) the crucial numeric properties of the implementation. They will be able to reject unsuitable implementations, and (possibly) to correctly characterize the accuracy of their own results. Programs will be able to detect (and possibly correct for) exceptions in arithmetic processing.

End users will find it easier to determine whether a (properly documented) application program is likely to execute satisfactorily on their platform. This can be done by comparing the documented requirements of the program against the documented properties of the platform.

Finally, end users of numeric application packages will be able to rely on the correct execution of those packages. That is, for correctly programmed algorithms, the results are reliable if and only if there is no notification.

Information technology — Language independent arithmetic —

Part 1: Integer and floating point arithmetic

1 Scope

This part of ISO/IEC 10967 specifies properties of many of the integer and floating point datatypes available in a variety of programming languages in common use for mathematical and numerical applications.

It is not the purpose of this part of ISO/IEC 10967 to ensure that an arbitrary numerical function can be so encoded as to produce acceptable results on all conforming datatypes. Rather, the goal is to ensure that the properties of the arithmetic on a conforming datatype are made available to the programmer. Therefore, it is not reasonable to demand that a substantive piece of software run on every implementation that can claim conformity to this part of ISO/IEC 10967.

An implementor may choose any combination of hardware and software support to meet the specifications of this part of ISO/IEC 10967. It is the datatypes and operations on values of those datatypes, of the computing environment as seen by the programmer/user, that does or does not conform to the specifications.

The term *implementation* (of this part of ISO/IEC 10967) denotes the total computing environment pertinent to this part of ISO/IEC 10967, including hardware, language processors, subroutine libraries, exception handling facilities, other software, and documentation.

1.1 Inclusions

This part of ISO/IEC 10967 provides specifications for properties of integer and floating point datatypes as well as basic operations on values of these datatypes. Specifications are included for bounded and unbounded integer datatypes, as well as floating point datatypes. Boundaries for the occurrence of exceptions and the maximum error allowed are prescribed for each specified operation. Also the result produced by giving a special value operand, such as an infinity or a NaN (not-a-number), is prescribed for each specified floating point operation.

This part of ISO/IEC 10967 provides specifications for:

- a) The set of required values of the arithmetic datatype.
- b) A number of arithmetic operations, including:
 - 1) comparison operations on two operands of the same type,
 - 2) primitive operations (addition, subtraction, etc.) with operands of the same type,
 - 3) operations that access properties of individual values,

- 4) conversion operations of a value from one arithmetic datatype to another arithmetic datatype, where at least one of the datatypes is conforming to this part of ISO/IEC 10967, and
- 5) numerals for all values specified by this part of ISO/IEC 10967 for a conforming datatype.

This part of ISO/IEC 10967 also provides specifications for:

- c) The results produced by an included floating point operation when one or more argument values are IEC 60559 special values.
- d) Program-visible parameters that characterise the values and certain aspects of the operations of an arithmetic datatype.
- e) Methods for reporting arithmetic exceptions.

1.2 Exclusions

This part of ISO/IEC 10967 provides no specifications for:

- a) Arithmetic and comparison operations whose operands are of more than one datatype. This part of ISO/IEC 10967 neither requires nor excludes the presence of such “mixed operand” operations.
- b) An interval datatype, or the operations on such data. This part of ISO/IEC 10967 neither requires nor excludes such data or operations.
- c) A fixed point datatype, or the operations on such data. This part of ISO/IEC 10967 neither requires nor excludes such data or operations.
- d) A rational datatype, or the operations on such data. This part of ISO/IEC 10967 neither requires nor excludes such data or operations.
- e) The properties of arithmetic datatypes that are not related to the numerical process, such as the representation of values on physical media.
- f) The properties of integer and floating point datatypes that properly belong in programming language standards or other specifications. Examples include:
 - 1) the syntax of numerals and expressions in the programming language, including the precedence of operators in the programming language,
 - 2) the syntax used for parsed (input) or generated (output) character string forms for numerals by any specific programming language or library,
 - 3) the presence or absence of automatic datatype coercions, and the consequences of applying an operation to values of improper type, or to uninitialised data,
 - 4) the rules for assignment, parameter passing, and returning value.

NOTE – See Clause 7 and Annex D for a discussion of language standards and language bindings.

The internal representation of values is beyond the scope of this standard. E.g., the value of the exponent bias, if any, is not specified, nor available as a parameter specified by this part

of ISO/IEC 10967. Internal representations need not be unique, nor is there a requirement for identifiable fields (for sign, exponent, and so on).

Furthermore, this part of ISO/IEC 10967 does not provide specifications for how the operations should be implemented or which algorithms are to be used for the various operations.

2 Conformity

It is expected that the provisions of this part of ISO/IEC 10967 will be incorporated by reference and further defined in other International Standards; specifically in programming language standards and in binding standards.

A binding standard specifies the correspondence between one or more of the arithmetic datatypes, parameters, and operations specified in this part of ISO/IEC 10967 and the concrete language syntax of some programming language. More generally, a binding standard specifies the correspondence between certain datatypes, parameters, and operations and the elements of some arbitrary computing entity. A language standard that explicitly provides such binding information can serve as a binding standard.

When a binding standard for a language exists, an implementation shall be said to conform to this part of ISO/IEC 10967 if and only if it conforms to the binding standard. In the case of conflict between a binding standard and this part of ISO/IEC 10967, the specifications of the binding standard takes precedence.

When a binding standard requires only a subset of the integer or floating point datatypes provided, an implementation remains free to conform to this part of ISO/IEC 10967 with respect to other datatypes independently of that binding standard.

When a binding standard requires only a subset of the operations specified in this part of ISO/IEC 10967, an implementation remains free to conform to this part of ISO/IEC 10967 with respect to other datatypes and operations, independently of that binding standard.

When no binding standard exists, an implementation conforms to this part of ISO/IEC 10967 if and only if it provides one or more datatypes and operations that together satisfy all the requirements of Clauses 5 through 8 that are relevant to those datatypes and operations. The implementation shall then document the binding.

Conformity to this part of ISO/IEC 10967 is always with respect to a specified set of datatypes and set of operations. Under certain circumstances, conformity to IEC 60559 is implied by conformity to this part of ISO/IEC 10967.

An implementation is free to provide arithmetic datatypes and arithmetic operations that do not conform to this part of ISO/IEC 10967 or that are beyond the scope of this part of ISO/IEC 10967. The implementation shall not claim conformity to this part of ISO/IEC 10967 for such datatypes or operations.

An implementation is permitted to have modes of operation that do not conform to this part of ISO/IEC 10967. A conforming implementation shall specify how to select the modes of operation that ensure conformity. However, a mode of operation that conforms to this part of ISO/IEC 10967 should be the default mode of operation.

NOTES

- 1 Language bindings are essential. Clause 8 requires an implementation to supply a binding if no binding standard exists. See Annex C.7 for recommendations on the proper content of a binding standard, Annex E for an example of a conformity statement, and Annex D for suggested language bindings.
- 2 A complete binding for this part of ISO/IEC 10967 may include (explicitly or by reference) a binding for IEC 60559 as well. See 5.2.1 and Annex B.
- 3 It is not possible to conform to this part of ISO/IEC 10967 without specifying to which datatypes and set of operations, and modes of operation, conformity is claimed.
- 4 This part of ISO/IEC 10967 requires that certain integer operations are made available for a conforming integer datatype, and that certain floating point operations are made available for a conforming floating point datatype.
- 5 All the operations specified in this part of ISO/IEC 10967 for a datatype must be provided for a conforming datatype, in a conforming mode of operation for that datatype.

3 Normative references

The following referenced documents are indispensable for the application of this part of ISO/IEC 10967. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 60559, *Standard for floating-point arithmetic*.

4 Symbols and definitions

4.1 Symbols

For the purposes of this document, the following symbols are used.

4.1.1 Operators and relations

All prefix and infix operators have their conventional exact mathematical meaning. In particular, this document uses:

$\Rightarrow$ and $\Leftrightarrow$ for logical implication and equivalence
 $+$, $-$, $/$, $|x|$, $\lfloor x \rfloor$, $\lceil x \rceil$, and $\text{round}(x)$ on real values
 $\cdot$ for multiplication on real values
 $<$, $\leq$, $\geq$, and $>$ between real values
 $=$ and $\neq$ between real as well as special values
 $\max$ on non-empty upwardly closed sets of real values
 $\min$ on non-empty downwardly closed sets of real values
 $\cup$, $\cap$, $\in$, $\notin$, $\subset$, $\subseteq$, $\not\subseteq$, $=$, and $\neq$ with sets
 $\times$ for the Cartesian product of sets
 $\rightarrow$ for a mapping between sets
 $|$ for the divides relation between integer values
 x^y , $\sqrt{x}$, $\log_b(x)$ on real values

NOTE 1 – $\approx$ is used informally, in notes and the rationale.

For $x \in \mathcal{R}$, the notation $\lfloor x \rfloor$ designates the largest integer not greater than x :

$$\lfloor x \rfloor \in \mathcal{Z} \quad \text{and} \quad x - 1 < \lfloor x \rfloor \leq x$$

the notation $\lceil x \rceil$ designates the smallest integer not less than x :

$$\lceil x \rceil \in \mathcal{Z} \quad \text{and} \quad x \leq \lceil x \rceil < x + 1$$

and the notation $\text{round}(x)$ designates the integer closest to x :

$$\text{round}(x) \in \mathcal{Z} \quad \text{and} \quad x - 0.5 \leq \text{round}(x) \leq x + 0.5$$

where in case x is exactly half-way between two integers, the even integer is the result.

The *divides* relation ($\mid$) on integers tests whether an integer i divides an integer j exactly:

$$i \mid j \Leftrightarrow (i \neq 0 \text{ and } i \cdot n = j \text{ for some } n \in \mathcal{Z})$$

NOTE 2 – $i \mid j$ is true exactly when j/i is defined and $j/i \in \mathcal{Z}$.

4.1.2 Sets and intervals

In this document, $\mathcal{Z}$ denotes the set of mathematical integers, $\mathcal{R}$ denotes the set of real numbers, and $\mathcal{C}$ denotes the set of complex numbers over $\mathcal{R}$. Note that $\mathcal{Z} \subset \mathcal{R} \subset \mathcal{C}$.

The conventional notation for set definition and for set operations are used.

The following notation for intervals is used in this document:

$[x, z]$ designates the interval $\{y \in \mathcal{R} \mid x \leq y \leq z\}$,

$]x, z]$ designates the interval $\{y \in \mathcal{R} \mid x < y \leq z\}$,

$[x, z[$ designates the interval $\{y \in \mathcal{R} \mid x \leq y < z\}$, and

$]x, z[$ designates the interval $\{y \in \mathcal{R} \mid x < y < z\}$.

NOTE – The notation using a round bracket for an open end of an interval is not used, for the risk of confusion with the notation for pairs.

4.1.3 Exceptional values

The parts of ISO/IEC 10967 use the following six exceptional values:

- a) **inexact**: the result is rounded and different from the exact result.
- b) **underflow**: the absolute value of the unrounded result is less than the smallest normal value, and the rounded result may have lost accuracy due to the denormalisation (more than lost by ordinary rounding if the exponent range was unbounded).
- c) **overflow**: the rounded result (when rounding as if the exponent range was unbounded) is larger than what can be represented in the result datatype.
- d) **infinitary**: the corresponding mathematical function has a pole at the finite argument point, or the result is otherwise infinite from finite arguments.

NOTE – **infinitary** is a generalisation of **divide_by_zero**.

- e) **invalid**: the operation is undefined but not infinitary, or the result is in $\mathcal{C}$ but not in $\mathcal{R}$, for the given arguments.

- f) **absolute_precision_underflow**: indicates that at least one argument is such that the density of representable values is too low in the neighbourhood of the given argument value for a numeric result to be considered appropriate to return. This exceptional value is used for operations that approximate trigonometric functions (Part 2 and Part 3) and for operations that approximate complex hyperbolic and exponentiation functions (Part 3).

For the exceptional values, a continuation value may be given in ISO/IEC 10967 in parenthesis after the exceptional value.

4.1.4 Special values

The following symbols represent special values defined in IEC 60559 and are used in ISO/IEC 10967:

-0, **+∞**, **-∞**, **qNaN**, and **sNaN**.

These values are not part of *I* or *F* (see Clauses 5.1 and 5.2 for a definition of these datatypes), but if

hasinf_I

(see Clause 5.1) has the value **true**, the **+∞**, **-∞** values are included in the integer datatype in the implementation that corresponds to *I*, and if *iec_60559_F* (see Clause 5.2.1) has the value **true**, all these special values are included in the floating point datatype in the implementation that corresponds to *F*.

NOTE – This document uses the above five special values for compatibility with IEC 60559.

In particular, the symbol **-0** (in bold) is not the application of (mathematical) unary **-** to the value 0, and is a value logically distinct from 0.

The specifications for floating point operations cover the results to be returned by an operation if given one or more of the IEC 60559 special values **-0**, **+∞**, **-∞**, or NaNs as input values. These specifications apply only to systems which provide and support these special values.

If an implementation is not capable of representing a **-0** result or continuation value, 0 shall be used as the actual result or continuation value. If an implementation is not capable of representing a prescribed result or continuation value of the IEC 60559 special values **+∞**, **-∞**, or **qNaN**, the actual result or continuation value is binding or implementation defined.

4.1.5 The Boolean datatype

The datatype **Boolean** consists of the two values **true** and **false**.

NOTE – Mathematical relations *are* true or false (or undefined, if an operand is undefined), which are abstract conditions, not values in a datatype. In contrast, **true** and **false** are values in **Boolean**.

4.1.6 Operation specification framework

Each of the operations are specified using a mathematical notation with cases. Each case condition is intended to be disjoint with the other cases, and encompass all non-special values as well as some of the special values.

Mathematically, each argument to an operation is a pair of a value and a set of exceptional values and likewise for the return value. However, in most cases only the first part of this pair is

written out in the specifications. The set of exceptional values returned from an operation is at least the union of the set of exceptional values from the arguments. Any new exceptional value that the operation itself gives rise to is given in the form **exceptional_value**(*continuation_value*) indicating that the second (implicit) part of the mathematical return value not only is the union of the second (implicit) parts of the arguments, but in addition is unioned with the singleton set of the given exceptional value, or, in the case of **underflow** or **overflow**, the set of the given exceptional value and **inexact**.

In an implementation, the exceptional values usually do not accompany each argument and return value, but are instead handled as notifications. See Clause 6.

When not communicating values, notifications shall be internal to each computational thread, whether threads are explicit or implicit in the program as seen by the programmer.

When communicating values, if the value sending thread has notifications that may be relevant for a communicated values these notifications should be communicated to a receiving thread along with values (of any datatype, not just numeric ones). In such instances, the exceptional values are associated with the value, even though it may pick up notifications in the thread that arose for a different computation in that thread and were not cleared.

NOTES

- 1 If notifications were arbitrarily seen in other threads, it would be very difficult to know which computation (thread) it is that might have caused the notification, and thus may trigger notification handling when not appropriate in an unrelated thread. Therefore it is essential that notifications are internal to each computational thread, when not communicating a value.
- 2 If notifications (normally recorded in indicators) are trimmed away when communicating a value (of whatever type) to another thread, that can result in the failure to cause notification handling when that would have been appropriate. Not communicating notifications between communicating threads thus goes against a goal set out in the introduction, namely “the executing program must be notified when proper results cannot be returned (e.g., when a computed result may be out of range or undefined)”.

However, many existing methods for remote procedure calling, or thread communication, do not communicate notifications (even when they are recorded in indicators).

4.2 Definitions of terms

For the purposes of this document, the following terms and definitions apply.

4.2.1

accuracy

closeness between the true mathematical result and a computed result

4.2.2

arithmetic datatype

datatype whose non-special values are members of $\mathcal{Z}$, $\mathcal{R}$, or $\mathcal{C}$

4.2.3

continuation value

computational value used as the result of an arithmetic operation when an exception occurs

Continuation values are intended to be used in subsequent arithmetic processing. A continuation value can be a (in the datatype representable) value in $\mathcal{R}$ or be an IEC 60559 special value. (Contrast with *exceptional value*. See Clause 6.2.1.)

4.2.4

denormalisation

inclusion of lead zero digits, with corresponding adjustment of the exponent

Denormalisation is logically done before rounding (otherwise there may be double rounding, that is rounding done twice with slightly different rounding functions, and that would be nonconforming). It may be done in order to get the exponent (just) within representable range.

4.2.5

denormalisation loss

larger than normal rounding error caused by the fact that denormalisation plus rounding may lose precision more than only rounding would do if the target exponent range was unbounded

See Clause 5.2.4 for a full definition.

4.2.6

error

⟨in computed value⟩ difference between a computed value and the mathematically correct value

Used in phrases like “rounding error” or “error bound”.

4.2.7

error

⟨computation gone awry⟩ exception

Used in phrases like “error message” or “error output”. Error and exception are not synonyms in any other contexts.

4.2.8

exception

inability of an operation to return a suitable finite numeric result from finite arguments

This might arise because no such finite result exists mathematically (**infinitary** (e.g., at a pole), **invalid** (e.g., when the true result is in $\mathcal{C}$ but not in $\mathcal{R}$)), or because the mathematical result cannot, or might not, be representable with sufficient accuracy (**underflow**, **overflow**) or viability (**absolute_precision_underflow**).

NOTES

- 1 **absolute_precision_underflow** is not used in this document, but is used in Part 2 (and thereby also in Part 3).
- 2 The term exception is here not used to designate certain methods of handling notifications that fall under the category ‘change of control flow’. Such methods of notification handling will be referred to as “[programming language name] exception”, when referred to, particularly in Annex D.

4.2.9

exceptional value

non-numeric value produced (in the specification model) by an arithmetic operation to indicate the occurrence of an exception (or the inexactness of the result)

Exceptional values are not used in subsequent arithmetic processing. (See Clause 5.)

NOTES

- 3 Exceptional values are used as a defining formalism only. With respect to this document, they do not represent values of any of the datatypes described. There is no requirement that they be represented or stored in the computing system.
- 4 Exceptional values are not to be confused with the NaNs and infinities defined in IEC 60559. Contrast this definition with that of *continuation value* above.

4.2.10

helper function

function used solely to aid in the expression of a requirement

Helper functions are not accessible to the programmer, and are not required to be part of an implementation.

4.2.11

implementation (of this document)

total arithmetic environment presented to a programmer, including hardware, language processors, exception handling facilities, subroutine libraries, other software, and all pertinent documentation

4.2.12

literal

single syntactic entity denoting a constant value

4.2.13

normal value

non-special and non-subnormal value of a floating point datatype F

See F_N in Clause 5.2 for a full definition.

4.2.14

notification

process by which a program (or that program's user) is informed that an arithmetic exception has occurred

For example, dividing 2 by 0 results in a notification for **infinitary**. See Clause 6 for details.

4.2.15

numeral

numeric literal

It may denote a value in $\mathcal{Z}$ or $\mathcal{R}$, **-0**, an infinity, or a NaN.

4.2.16

operation

function that is intended to be made directly available to the programmer

As opposed to helper functions or theoretical mathematical functions.

4.2.17

pole

argument, x_0 , where a given mathematical function, f , is defined, finite, monotone, and continuous in at least one one path of approach towards x_0 , and where $\lim_{x \rightarrow x_0} f(x)$ is infinite

4.2.18

precision

number of digits in the fraction of a floating point number

(See Clause 5.2.)

4.2.19

rounding

act of computing a result for an operation that is close to the exact result for that operation, but that does not have digits beyond what the target datatype can represent

Note that a suitable representable result may not exist (see Clause 5.2.5).

4.2.20

rounding function

function, $rnd : \mathcal{R} \rightarrow X$, (where X is a given discrete and unlimited subset of $\mathcal{R}$) that maps each element of X to itself, and is monotonic non-decreasing

Formally, if x and y are in $\mathcal{R}$,

$$\begin{aligned}x \in X &\Rightarrow rnd(x) = x \\x < y &\Rightarrow rnd(x) \leq rnd(y)\end{aligned}$$

Note that if u is between two adjacent values in X , $rnd(u)$ selects one of those adjacent values.

4.2.21

round to nearest

rounding function, rnd , that when $u \in \mathcal{R}$ is strictly between two adjacent values in X , $rnd(u)$ selects the one nearest u , but if the adjacent values are equidistant from u , either value can be chosen deterministically but in such a way that sign symmetry is preserved ($rnd(-u) = -rnd(u)$)

4.2.22

round toward minus infinity

rounding function, rnd , that when $u \in \mathcal{R}$ is strictly between two adjacent values in X , $rnd(u)$ selects the one less than u

4.2.23**round toward plus infinity**

rounding function, rnd , that when $u \in \mathcal{R}$ is strictly between two adjacent values in X , $rnd(u)$ selects the one greater than u

4.2.24**signature** (of a function or operation)

argument and result summary of information about an operation or function

A signature includes the function or operation name; a subset of allowed argument values to the operation; and a superset of results from the function or operation (including exceptional values if any), if the argument is in the subset of argument values given in the signature.

The signature $add_I : I \times I \rightarrow I \cup \{\mathbf{overflow}\}$ states that the operation named add_I shall accept any pair of values in I as input, and when given such input shall return either a single value in I as its output or the exceptional value **overflow** possibly accompanied by a continuation value.

A signature for an operation or function does not forbid the operation from accepting a wider range of arguments, nor does it guarantee that every value in the result range will actually be returned for some argument(s). An operation given an argument outside the stipulated argument domain may produce a result outside the stipulated result range.

NOTE 5 – In particular, IEC 60559 special values are not in F , but must be accepted as arguments if $iec-60559_F$ has the value **true**.

4.2.25**subnormal****denormal** (obsolete)

value of a floating point datatype F , or $-\mathbf{0}$, whose absolute value is strictly less than the smallest positive normal value in F ($fminN_F$)

(See F_S in Clause 5.2 for a full definition.)

4.2.26**ulp****unit(s) in the last place** (for a given real value and given floating point datatype)

for a value x in $\mathcal{R}$, that has a nearest-closer-to-zero normalised value in F extended to arbitrarily large values, where the normalised value's exponent is t , precision is p_F , and the radix is r_F : the unit is $r_F^{t-p_F}$; for a value x in $\mathcal{R}$, with a nearest-closer-to-zero subnormal value in F , as well as for $-\mathbf{0}$: the unit is $fminD_F$

This value depends on the exponent, the radix, and the precision used in representing the numbers in F . (See Clause 5.2.)

NOTE 6 – For a value that is exactly equal to an integer power of the radix, the ulp is the size of the gap between available values on the side *away* from zero.

5 Specifications for integer and floating point datatypes and operations

An arithmetic datatype consists of a set of values and is accompanied by operations that take values from an arithmetic datatype and return a value in an arithmetic datatype or a boolean value. For any particular arithmetic datatype, the set of non-special values is characterised by a small number of parameters. An exact definition of the value set will be given in terms of these parameters.

Each operation is given a signature and is further specified by a number of cases. These cases may refer to mathematical functions, to other operations, and to helper functions (specified in this document). They also use special values and exceptional values.

Given the datatype's non-special value set, V , the accompanying arithmetic operations will be specified as mathematical functions on V union certain special values that may be in the corresponding implementation datatype. These functions typically return values in V or a special value, but they may instead nominally return exceptional values (that have no arithmetic datatype, and are not to be confused with the special values) that are often specified along with a continuation value. Though nominally listed as a return value, an exceptional value is mathematically really part of a second component of the result, as explained in clause 4.1.6, and is to be handled as a notification as described in clause 6.

The exceptional values used in this document are **underflow**, **inexact**, **overflow**, **infinitary** (generalisation of division-by-zero), and **invalid**. Parts 2 and 3 will also use the exceptional value **absolute_precision_underflow** for the operations that correspond to cyclic functions. For many cases this document specifies which continuation value to use with a specified exceptional value. The continuation value is then expressed in parenthesis after the expression of the exceptional value. For example, **infinitary**($+\infty$) expresses that the exceptional value **infinitary** in that case is to be accompanied by a continuation value of $+\infty$ (unless the binding states differently). In case the notification is by recording in indicators (see Clause 6.2.1), the continuation value is used as the actual return value. This part of ISO/IEC 10967 sometimes leaves the continuation value unspecified, in which case the continuation value is implementation defined.

Whenever an arithmetic operation (as defined in this clause) returns an exceptional value (mathematically, that a non-empty exceptional value set is unioned with the union of exceptions from the arguments, as the exceptional values part of the result), notification of this shall occur as described in Clause 6.

An implementation of a conforming integer or floating point datatype shall include all non-special values defined for that datatype by this document. However, the implementing datatype is permitted to include additional values (for example, and in particular, IEC 60559 special values). This part of ISO/IEC 10967 specifies the behaviour of integer operations when applied to infinitary values, but not for other such additional values. This part of ISO/IEC 10967 specifies the behaviour of floating point operations when applied to IEC 60559 special values, but not for other such additional values.

An implementation of a conforming integer or floating point datatype shall be accompanied by all the operations specified for that datatype by this part of ISO/IEC 10967. Additional operations are explicitly permitted.

The datatype **Boolean** is used for parameters and the results of comparison operations. An implementation is not required by this document to provide a **Boolean** datatype, nor is it re-

quired by this part of ISO/IEC 10967 to provide operations on **Boolean** values. However, an implementation shall provide a method of distinguishing **true** from **false** as parameter values and as results of operations.

NOTE – This document requires an implementation to provide methods to access values, operations, and other facilities. Ideally, these methods are provided by a language or binding standard, and the implementation merely cites this standard. Only if a binding standard does not exist, must an individual implementation supply this information on its own. See Annex C.7.

5.1 Integer datatypes and operations

The non-special value set, I , for an integer datatype shall be a subset of $\mathcal{Z}$, characterised by the following parameters:

$bounded_I \in \mathbf{Boolean}$ (whether the set I is finite)
 $minint_I \in I \cup \{-\infty\}$ (the smallest integer in I if $bounded_I = \mathbf{true}$)
 $maxint_I \in I \cup \{+\infty\}$ (the largest integer in I if $bounded_I = \mathbf{true}$)

In addition, the following parameter characterises one aspect of the special values in the datatype corresponding to I in the implementation:

$hasinf_I \in \mathbf{Boolean}$ (whether the corresponding datatype has $-\infty$ and $+\infty$)

NOTE 1 – The first edition of this document also specified the parameter $modulo_I$. A binding may still have a parameter $modulo_I$, and for conformity to this second edition, that parameter is to have the value **false**. Part 2 includes specifications for operations add_wrap_I , sub_wrap_I , and mul_wrap_I . If the parameter $modulo_I$ has the value **true** (non-conforming case), that indicates that the binding binds the basic integer arithmetic operations, for bounded integer datatypes, to the corresponding wrapping operations instead of the add_I , sub_I , and mul_I operations of this document.

If $bounded_I$ is **false**, the set I shall satisfy

$$I = \mathcal{Z}$$

In this case, $hasinf_I$ shall be **true**, the value of $minint_I$ shall be $-\infty$, and the value of $maxint_I$ shall be $+\infty$.

If $bounded_I$ is **true**, then $minint_I \in \mathcal{Z}$ and $maxint_I \in \mathcal{Z}$ and the set I shall satisfy

$$I = \{x \in \mathcal{Z} \mid minint_I \leq x \leq maxint_I\}$$

and $minint_I$ and $maxint_I$ shall satisfy

$$maxint_I > 0$$

and one of:

$$\begin{aligned} minint_I &= 0, \\ minint_I &= -maxint_I, \text{ or} \\ minint_I &= -(maxint_I + 1) \end{aligned}$$

A bounded integer datatype with $minint_I < 0$ is called *signed*. A bounded integer datatype with $minint_I = 0$ is called *unsigned*. An integer datatype in which $bounded_I$ is **false** is *signed*, due to the requirement above.

An implementation may provide more than one integer datatype. A method shall be provided for a program to obtain the values of the parameters $bounded_I$, $hasinf_I$, $minint_I$, and $maxint_I$, for each conforming integer datatype provided.

NOTES

- 2 The value of $hasinf_I$ does not affect the values of $minint_I$ and $maxint_I$ for bounded integer datatypes.
- 3 Most traditional programming languages call for bounded integer datatypes. Others allow or require an integer datatype to have an unbounded range. A few languages permit the implementation to decide whether an integer datatype will be bounded or unbounded. (See C.5.1.0.1 for further discussion.)
- 4 Operations on unbounded integers will not overflow, but may fail due to exhaustion of resources.
- 5 Unbounded natural numbers are not covered by this document.

5.1.1 Integer result function

If $bounded_I$ is **true**, the mathematical operations $+$, $-$, and $\cdot$ can produce results that lie outside the set I even when given values in I . In such cases, the computational operations add_I , sub_I , neg_I , abs_I , and mul_I shall cause an overflow notification.

In the integer operation specifications below, the handling of overflow is specified via the $result_I$ helper function:

$$result_I : \mathcal{Z} \rightarrow I \cup \{\mathbf{overflow}\}$$

which is defined by:

$$\begin{aligned} result_I(x) &= x && \text{if } x \in I \\ &= \mathbf{overflow}(-\infty) && \text{if } x \in \mathcal{Z} \text{ and } x \notin I \text{ and } x < 0 \\ &= \mathbf{overflow}(+\infty) && \text{if } x \in \mathcal{Z} \text{ and } x \notin I \text{ and } x > 0 \end{aligned}$$

NOTES

- 1 For integer operations, this document does not specify continuation values for **overflow** when $hasinf_I = \mathbf{false}$ nor the continuation values for **invalid**. The binding or implementation must document the continuation value(s) used for such cases (see Clause 8).
- 2 For the floating point operations in Clause 5.2 a $result_F$ helper function is used to consistently and succinctly express overflow and denormalisation loss cases.

5.1.2 Integer operations

5.1.2.1 Comparisons

For each provided conforming integer datatype, the following operations shall be provided.

$$eq_I : I \times I \rightarrow \mathbf{Boolean}$$

$$\begin{aligned} eq_I(x, y) &= \mathbf{true} && \text{if } x, y \in I \cup \{-\infty, +\infty\} \text{ and } x = y \\ &= \mathbf{false} && \text{if } x, y \in I \cup \{-\infty, +\infty\} \text{ and } x \neq y \end{aligned}$$

$$neg_I : I \times I \rightarrow \mathbf{Boolean}$$

$$\begin{aligned} neg_I(x, y) &= \mathbf{true} && \text{if } x, y \in I \cup \{-\infty, +\infty\} \text{ and } x \neq y \\ &= \mathbf{false} && \text{if } x, y \in I \cup \{-\infty, +\infty\} \text{ and } x = y \end{aligned}$$

$lss_I : I \times I \rightarrow \text{Boolean}$

$$\begin{aligned} lss_I(x, y) &= \text{true} \\ &= \text{false} \\ &= \text{true} \\ &= \text{true} \\ &= \text{false} \\ &= \text{false} \end{aligned}$$

$$\begin{aligned} &\text{if } x, y \in I \text{ and } x < y \\ &\text{if } x, y \in I \text{ and } x \geq y \\ &\text{if } x \in I \cup \{-\infty\} \text{ and } y = +\infty \\ &\text{if } x = -\infty \text{ and } y \in I \\ &\text{if } x \in I \cup \{-\infty, +\infty\} \text{ and } y = -\infty \\ &\text{if } x = +\infty \text{ and } y \in I \cup \{+\infty\} \end{aligned}$$
 $leq_I : I \times I \rightarrow \text{Boolean}$

$$\begin{aligned} leq_I(x, y) &= \text{true} \\ &= \text{false} \\ &= \text{true} \\ &= \text{true} \\ &= \text{false} \\ &= \text{false} \end{aligned}$$

$$\begin{aligned} &\text{if } x, y \in I \text{ and } x \leq y \\ &\text{if } x, y \in I \text{ and } x > y \\ &\text{if } x \in I \cup \{-\infty, +\infty\} \text{ and } y = +\infty \\ &\text{if } x = -\infty \text{ and } y \in I \cup \{-\infty\} \\ &\text{if } x \in I \cup \{+\infty\} \text{ and } y = -\infty \\ &\text{if } x = +\infty \text{ and } y \in I \end{aligned}$$
 $gtr_I : I \times I \rightarrow \text{Boolean}$
 $gtr_I(x, y) = lss_I(y, x)$
 $geq_I : I \times I \rightarrow \text{Boolean}$
 $geq_I(x, y) = leq_I(y, x)$

5.1.2.2 Basic arithmetic

For each provided conforming integer datatype, the following operations shall be provided. If I is unsigned, it is permissible to omit the operations neg_I , abs_I , and $signum_I$.

 $neg_I : I \rightarrow I \cup \{\text{overflow}\}$

$$\begin{aligned} neg_I(x) &= result_I(-x) \\ &= +\infty \\ &= -\infty \end{aligned}$$

$$\begin{aligned} &\text{if } x \in I \\ &\text{if } x = -\infty \\ &\text{if } x = +\infty \end{aligned}$$
 $add_I : I \times I \rightarrow I \cup \{\text{overflow}\}$

$$\begin{aligned} add_I(x, y) &= result_I(x + y) \\ &= -\infty \\ &= -\infty \\ &= +\infty \\ &= +\infty \\ &= \text{invalid} \\ &= \text{invalid} \end{aligned}$$

$$\begin{aligned} &\text{if } x, y \in I \\ &\text{if } x \in I \cup \{-\infty\} \text{ and } y = -\infty \\ &\text{if } x = -\infty \text{ and } y \in I \\ &\text{if } x \in I \cup \{+\infty\} \text{ and } y = +\infty \\ &\text{if } x = +\infty \text{ and } y \in I \\ &\text{if } x = +\infty \text{ and } y = -\infty \\ &\text{if } x = -\infty \text{ and } y = +\infty \end{aligned}$$
 $sub_I : I \times I \rightarrow I \cup \{\text{overflow}\}$

$$\begin{array}{lll}
 \text{sub}_I(x, y) & = \text{result}_I(x - y) & \text{if } x, y \in I \\
 & = -\infty & \text{if } x \in I \cup \{-\infty\} \text{ and } y = +\infty \\
 & = -\infty & \text{if } x = -\infty \text{ and } y \in I \\
 & = +\infty & \text{if } x \in I \cup \{+\infty\} \text{ and } y = -\infty \\
 & = +\infty & \text{if } x = +\infty \text{ and } y \in I \\
 & = \text{invalid} & \text{if } x = +\infty \text{ and } y = +\infty \\
 & = \text{invalid} & \text{if } x = -\infty \text{ and } y = -\infty
 \end{array}$$

$$\begin{array}{lll}
 \text{mul}_I : I \times I \rightarrow I \cup \{\text{overflow}\} & & \\
 \text{mul}_I(x, y) & = \text{result}_I(x \cdot y) & \text{if } x, y \in I \\
 & = +\infty & \text{if } x = +\infty \text{ and } (y = +\infty \text{ or } (y \in I \text{ and } y > 0)) \\
 & = -\infty & \text{if } x = +\infty \text{ and } (y = -\infty \text{ or } (y \in I \text{ and } y < 0)) \\
 & = -\infty & \text{if } x \in I \text{ and } x > 0 \text{ and } y = -\infty \\
 & = +\infty & \text{if } x \in I \text{ and } x < 0 \text{ and } y = -\infty \\
 & = +\infty & \text{if } x = -\infty \text{ and } (y = -\infty \text{ or } (y \in I \text{ and } y < 0)) \\
 & = -\infty & \text{if } x = -\infty \text{ and } (y = +\infty \text{ or } (y \in I \text{ and } y > 0)) \\
 & = -\infty & \text{if } x \in I \text{ and } x < 0 \text{ and } y = +\infty \\
 & = +\infty & \text{if } x \in I \text{ and } x > 0 \text{ and } y = +\infty \\
 & = \text{invalid} & \text{if } x \in \{-\infty, +\infty\} \text{ and } y = 0 \\
 & = \text{invalid} & \text{if } x = 0 \text{ and } y \in \{-\infty, +\infty\}
 \end{array}$$

$$\begin{array}{lll}
 \text{abs}_I : I \rightarrow I \cup \{\text{overflow}\} & & \\
 \text{abs}_I(x) & = \text{result}_I(|x|) & \text{if } x \in I \\
 & = +\infty & \text{if } x \in \{-\infty, +\infty\}
 \end{array}$$

$$\begin{array}{lll}
 \text{signum}_I : I \rightarrow \{-1, 1\} & & \\
 \text{signum}_I(x) & = 1 & \text{if } (x \in I \text{ and } x \geq 0) \text{ or } x = +\infty \\
 & = -1 & \text{if } (x \in I \text{ and } x < 0) \text{ or } x = -\infty
 \end{array}$$

NOTE 1 – The first edition of this document specified a slightly different operation sign_I . signum_I is consistent with signum_F , which in turn is consistent with the branch cuts for the complex trigonometric operations (Part 3).

Integer division with floor and its remainder:

$$\begin{array}{lll}
 \text{quot}_I : I \times I \rightarrow I \cup \{\text{overflow}, \text{infinitary}, \text{invalid}\} & & \\
 \text{quot}_I(x, y) & = \text{result}_I(\lfloor x/y \rfloor) & \text{if } x, y \in I \text{ and } y \neq 0 \\
 & = \text{infinitary}(+\infty) & \text{if } x \in I \text{ and } x > 0 \text{ and } y = 0 \\
 & = \text{infinitary}(-\infty) & \text{if } x \in I \text{ and } x < 0 \text{ and } y = 0 \\
 & = 0 & \text{if } x \in I \text{ and } y \in \{-\infty, +\infty\} \\
 & = \text{mul}_I(x, y) & \text{if } x \in \{-\infty, +\infty\} \text{ and } y \in I \text{ and } y \neq 0 \\
 & = \text{invalid} & \text{otherwise}
 \end{array}$$

NOTE 2 – $\text{quot}_I(\text{minint}_I, -1)$, for a bounded signed integer datatype where $\text{minint}_I = -\text{maxint}_I - 1$, is the only case where this operation will overflow.

$$\begin{aligned}
\text{mod}_I : I \times I &\rightarrow I \cup \{\text{invalid}\} \\
\text{mod}_I(x, y) &= x - (\lfloor x/y \rfloor \cdot y) && \text{if } x, y \in I \text{ and } y \neq 0 \\
&= x && \text{if } x \in I \text{ and } y \in \{-\infty, +\infty\} \\
&= \text{invalid} && \text{otherwise}
\end{aligned}$$

NOTES

- 3 The first edition of this document specified the operations div_I^f , div_I^t , mod_I^a , mod_I^p , rem_I^f , and rem_I^t . However, $\text{div}_I^f = \text{quot}_I$, and $\text{mod}_I^a = \text{rem}_I^f = \text{mod}_I$. Further, div_I^t , mod_I^p , and rem_I^t are not recommended to be provided, as their use may give rise to late-discovered bugs.
- 4 Part 2 specifies the related operations ratio_I , residue_I , group_I , and pad_I .

5.2 Floating point datatypes and operations

A floating point datatype shall have a non-special value set F that is a finite subset of $\mathcal{R}$, characterized by the following parameters:

$$\begin{aligned}
r_F \in \mathbb{Z} & \quad (\text{the radix of } F) \\
p_F \in \mathbb{Z} & \quad (\text{the precision of } F) \\
\text{emax}_F \in \mathbb{Z} & \quad (\text{the largest exponent of } F) \\
\text{emin}_F \in \mathbb{Z} & \quad (\text{the smallest exponent of } F) \\
\text{denorm}_F \in \mathbf{Boolean} & \quad (\text{whether } F \text{ contains non-zero subnormal values})
\end{aligned}$$

In addition, the following parameter characterises the special values in the datatype corresponding to F in the implementation, and the operations in common for this document and IEC 60559:

$$\text{iec_60559}_F \in \mathbf{Boolean} \quad (\text{whether the datatype and operations conform to IEC 60559})$$

NOTES

- 1 This standard does not advocate any particular representation for floating point values. However, concepts such as *radix*, *precision*, and *exponent* are derived from an abstract model of such values as discussed in Annex C.5.2.
- 2 The 2011 version of IEC 60559 also uses the parameters *emax* and *emin* (written as E_{max} and E_{min} in the 1989 version). However, those values are respectively one less than the emax_F and emin_F parameters of this document. The latter are, however, in line with the maximum and minimum exponent access variables in several programming languages.

The parameters r_F , p_F , and denorm_F shall satisfy:

$$\begin{aligned}
r_F &\geq 2 \\
p_F &\geq 2 \cdot \max\{1, \lceil \log_{r_F}(2 \cdot \pi) \rceil\} \\
\text{denorm}_F &= \mathbf{true}
\end{aligned}$$

NOTE 3 – The first edition of this document only required for p_F that $p_F \geq 2$. The requirement in this edition allows for the use of any floating point type in widespread use and is made so that angles in radians are not too degenerate within the first two cycles, plus and minus, when represented in F .

Furthermore, r_F should be even, and p_F should be such that $p_F \geq 2 + \lceil \log_{r_F}(1000) \rceil$.

NOTE 4 – The recommendation that $p_F \geq 2 + \lceil \log_{r_F}(1000) \rceil$, which did not occur in the first edition of this document, allows for the use of any floating point type in widespread use and is made so as to allow for a not too coarse angle resolution, for operations in Part 2 and Part 3, anywhere in the interval $[-\text{big_angle}_{r_F}, \text{big_angle}_{r_F}]$ (big_angle_{r_F} is a parameter introduced in Part 2).

The parameters $emin_F$ and $emax_F$ shall satisfy:

$$\begin{aligned} 1 - r_F^{p_F} &\leq emin_F \leq -1 - p_F \\ p_F &\leq emax_F \leq r_F^{p_F} - 1 \end{aligned}$$

and should satisfy:

$$0 \leq emax_F + emin_F \leq 4$$

NOTE 5 – The first edition of this document had the wider range requirement $1 - r_F^{p_F} \leq emin_F \leq 2 - p_F$. The shorter range requirement in this edition of this document allows for the use of any floating point type in widespread use and is made so as to be able to avoid the underflow notification, that is, avoid denormalisation loss, in the specifications for the $expm1_F$ and $ln1p_F$ operations (Part 2) for subnormal arguments (though these operations are still inexact for non-zero subnormal arguments).

Given specific values for r_F , p_F , $emin_F$, $emax_F$, and $denorm_F$, the following sets are defined:

$$F_S = \{s \cdot m \cdot r_F^{e-p_F} \mid s \in \{-1, 1\}, m, e \in \mathbb{Z}, 0 \leq m < r_F^{p_F-1}, e = emin_F\}$$

$$F_N = \{s \cdot m \cdot r_F^{e-p_F} \mid s \in \{-1, 1\}, m, e \in \mathbb{Z}, r_F^{p_F-1} \leq m < r_F^{p_F}, emin_F \leq e \leq emax_F\}$$

$$F_E = \{s \cdot m \cdot r_F^{e-p_F} \mid s \in \{-1, 1\}, m, e \in \mathbb{Z}, r_F^{p_F-1} \leq m < r_F^{p_F}, emax_F < e\}$$

$$F_L = \{s \cdot m \cdot r_F^{e-p_F} \mid s \in \{-1, 1\}, m, e \in \mathbb{Z}, r_F^{p_F-1} \leq m < r_F^{p_F}, e < emin_F\}$$

$$F^\dagger = F_S \cup F_N \cup F_E$$

$$F^\ddagger = F_L \cup F^\dagger$$

$$\begin{aligned} F &= F_S \cup F_N && \text{if } denorm_F = \mathbf{true} \\ &= \{0\} \cup F_N && \text{if } denorm_F = \mathbf{false} \quad (\text{non-conforming case, see Annex A}) \end{aligned}$$

NOTES

- 6 $F^\dagger$ is the outwards unbounded extension of F , including in addition all subnormal values that would be in F if $denorm_F$ were **true**. $F^\dagger$ will be used in defining rounding for operations.
- 7 $F^\ddagger$ is the unbounded extension of F .

The elements of F_N are called *normal* floating point values. The elements of F_S , as well as the special value $-\mathbf{0}$, are called *subnormal* floating point values.

NOTE 8 – The terms *normal* and *subnormal* refer to the mathematical values involved, not to any method of representation.

An implementation may provide more than one floating point datatype.

For each of the parameters r_F , p_F , $emin_F$, $emax_F$, $denorm_F$, and iec_60559_F , and for each conforming floating point datatype provided, a method shall be provided for a program to obtain the value of the parameter.

NOTE 9 – The conditions placed upon the parameters r_F , p_F , $emin_F$, and $emax_F$ are sufficient to guarantee that the abstract model of F is well-defined and contains its own parameters, as well as enabling the avoidance of denormalisation loss (in particular for $expm1_F$ and $ln1p_F$ of Part 2). More stringent conditions are needed to produce a computationally useful floating point datatype. These are design decisions which are beyond the scope of this document. (See Annex C.5.2.)

5.2.1 Conformity to IEC 60559

The parameter *iec_60559_F* shall be true only when the datatype corresponding to *F* and the relevant operations completely conform to the requirements of IEC 60559. *F* may correspond to any of the floating point datatypes defined in IEC 60559.

When *iec_60559_F* has the value **true**, all the facilities required by IEC 60559 shall be provided. Methods shall be provided for a program to access each such facility. In addition, documentation shall be provided to describe these methods, and all implementation choices. When *iec_60559_F* has the value **true**, all operations and values common to this document and IEC 60559 shall satisfy the requirements of both standards.

NOTES

- 1 IEC 60559 is also known as IEEE 754 [34].
- 2 The IEC 60559 facilities include: values for infinities and NaNs, extended comparisons, rounding towards positive or negative infinity, an exceptions (including inexact) recorded in indicators. See annex B for more information.
- 3 IEC 60559, third edition, specifies $r_F = 2$ or $r_F = 10$, as well as values for maximum and minimum exponents and precision for the floating point datatypes it specifies.
- 4 If *iec_60559_F* is **true**, then *denorm_F* must also be **true**. Note that *denorm_F* = **false** is non-conforming also to this document.

5.2.2 Range and granularity constants

The range and granularity of *F* is characterized by the following derived constants:

$$\begin{aligned}
 fmax_F &= \max F &&= (1 - r_F^{-p_F}) \cdot r_F^{emax_F} \\
 fminN_F &= \min \{z \in F_N \mid z > 0\} = r^{emin_F - 1} \\
 fminD_F &= \min \{z \in F_S \mid z > 0\} = r^{emin_F - p_F} \\
 fmin_F &= \min \{z \in F \mid z > 0\} = fminD_F \text{ if } denorm_F = \mathbf{true} \\
 &= fminN_F \text{ if } denorm_F = \mathbf{false} \text{ (non-conforming case)} \\
 epsilon_F &= r_F^{1-p_F} && \text{(the relative spacing in } F^\dagger \text{ between adjacent values)}
 \end{aligned}$$

For each of the derived constants *fmax_F*, *fminN_F*, *fmin_F*, and *epsilon_F*, and for each conforming floating point datatype provided, a method shall be provided for a program to obtain the value of the derived constant.

5.2.3 Approximate operations

The operations (specified below) *add_F*, *sub_F*, *mul_F*, *div_F* and, upon denormalisation loss, *scale_{F,I}* are approximations of exact mathematical operations. They differ from their mathematical counterparts, not only in that they may accept special values as arguments, but also in that

- a) they may produce “rounded” results,
- b) they may produce a special value (even without notification, or for values in *F* as arguments), and

c) they may produce notifications (with values in F or special values as continuation values).

The approximate floating point operations are specified *as if* they were computed in three stages:

- 1) compute the exact mathematical answer (if there is any), or determine if an **infinitary** or **invalid** notification is required,
- 2) round the exact answer (if there is any) to p_F digits of precision in the radix r_F (the precision will be less if the rounded answer is subnormal), maybe producing a **−0** as the rounded result, and
- 3) determine if an **underflow**, **overflow** (using $+\infty$ or $-\infty$ as continuation value if available), or **inexact** notification is required.

These stages will be modelled by basic and elementary mathematical functions (stage 1) and two helper functions: $nearest_F$ (stage 2) and $result_F$ (stage 3). These helper functions are not visible to the programmer and are not required to be part of the implementation, just like exact mathematical functions are not required to be part of an implementation. An actual implementation need not perform the above stages at all, merely return a result (or produce a notification and a continuation value) as if it had.

5.2.4 Rounding and rounding constants

Define the helper function $e_F : \mathcal{R} \rightarrow \mathcal{Z}$ such that

$$\begin{aligned} e_F(x) &= \lfloor \log_{r_F}(|x|) \rfloor + 1 && \text{if } |x| \geq fminN_F \\ &= emin_F && \text{if } |x| < fminN_F \end{aligned}$$

Define the helper function $u_F : \mathcal{R} \rightarrow F^\dagger$ such that

$$u_F(x) = r_F^{e_F(x) - p_F}$$

NOTES

- 1 The value $e_F(x)$ is the exponent for values in $F^\dagger$ in the immediate neighbourhood of x (which need not be in $F^\dagger$). When x is in $] -r_F \cdot fminN_F, r_F \cdot fminN_F[$, then $e_F(x)$ is $emin_F$ regardless of x .
- 2 The value $u_F(x)$ is the absolute distance between adjacent values in $F^\dagger$ in the immediate neighbourhood of x (which need not be in $F^\dagger$). When x is in $] -r_F \cdot fminN_F, r_F \cdot fminN_F[$, then $u_F(x)$ is $fminD_F$ regardless of x .
- 3 When x is on an exponent boundary of $F^\dagger$, the neighbourhood (mentioned in the previous two notes) is that which is away from zero.

For floating point operations, rounding is the process of taking an exact result in $\mathcal{R}$ and producing a p_F -digit approximation.

NOTE 4 – In Annex A of this document, and in Parts 2 and 3 of ISO/IEC 10967, the “exact result” may be a prerounding approximation, through approximation helper functions.

The $nearest_F$, $down_F$, and up_F helper functions are introduced to model the rounding process: The floating point helper function

$$nearest_F : \mathcal{R} \rightarrow F^\dagger$$

is the rounding function that rounds to nearest, ties rounded to even last digit. The floating point helper function

$$down_F : \mathcal{R} \rightarrow F^\dagger$$

is the rounding function that rounds towards negative infinity. The floating point helper function

$$up_F : \mathcal{R} \rightarrow F^\dagger$$

is the rounding function that rounds towards positive infinity.

If, for some $x \in \mathcal{R}$ and some $i \in \mathcal{Z}$, such that $|x| < fminN_F$, $|x \cdot r_F^i| \geq fminN_F$, and rounding function $rnd : \mathcal{R} \rightarrow F^\dagger$, the formula

$$rnd(x \cdot r_F^i) = rnd(x) \cdot r_F^i$$

does not hold, then rnd is said to have a *denormalisation loss* at x .

NOTE 5 – If $nearest'_F : \mathcal{R} \rightarrow F^\dagger$ is a round to nearest function to $F^\dagger$, and $nearest'_F(x) \neq nearest_F(x)$, then $nearest_F$ has a denormalisation loss at x . Similarly for up_F and $down_F$.

5.2.5 Floating point result function

A floating point operation produces a rounded result or a notification. The decision is based on the computed result (either before or after rounding).

The $result_F$ helper function is introduced to model this decision. $result_F$ is partially implementation defined. $result_F$ has a signature:

$$result_F : \mathcal{R} \times (\mathcal{R} \rightarrow F^\dagger) \rightarrow F \cup \{\mathbf{inexact}, \mathbf{underflow}, \mathbf{overflow}\}$$

NOTE 1 – The first input to $result_F$ is the computed result before rounding, and the second input is the rounding function to be used.

For the overflow cases for the three roundings $nearest_F$, up_F , and $down_F$, and for $x \in \mathcal{R}$, the following shall apply:

$$\begin{aligned} result_F(x, nearest_F) &= \mathbf{overflow}(+\infty) && \text{if } nearest_F(x) > fmax_F \\ result_F(x, nearest_F) &= \mathbf{overflow}(fmax_F) \text{ or } \mathbf{inexact}(fmax_F) && \text{if } nearest_F(x) = fmax_F \text{ and } x > fmax_F \\ result_F(x, nearest_F) &= \mathbf{overflow}(-\infty) && \text{if } nearest_F(x) < -fmax_F \\ result_F(x, nearest_F) &= \mathbf{overflow}(-fmax_F) \text{ or } \mathbf{inexact}(-fmax_F) && \text{if } nearest_F(x) = -fmax_F \text{ and } x < -fmax_F \\ result_F(x, up_F) &= \mathbf{overflow}(+\infty) && \text{if } up_F(x) > fmax_F \\ result_F(x, up_F) &= \mathbf{overflow}(-fmax_F) && \text{if } up_F(x) < -fmax_F \\ result_F(x, up_F) &= \mathbf{overflow}(-fmax_F) \text{ or } \mathbf{inexact}(-fmax_F) && \text{if } up_F(x) = -fmax_F \text{ and } x < -fmax_F \\ result_F(x, down_F) &= \mathbf{overflow}(-\infty) && \text{if } down_F(x) < -fmax_F \\ result_F(x, down_F) &= \mathbf{overflow}(fmax_F) && \text{if } down_F(x) > fmax_F \\ result_F(x, down_F) &= \mathbf{overflow}(fmax_F) \text{ or } \mathbf{inexact}(fmax_F) && \text{if } down_F(x) = fmax_F \text{ and } x > fmax_F \end{aligned}$$

For other cases for either of the three rounding functions as rnd , and for $x \in [-fmax_F, fmax_F]$, the following shall apply:

$$\begin{aligned} result_F(x, rnd) &= x && \text{if } x \in F \\ &= \mathbf{inexact}(rnd(x)) && \text{if } x \neq rnd(x) \text{ and } fminN_F \leq |x| \leq fmax_F \\ &= \mathbf{underflow}(-0) && \text{if } denorm_F = \mathbf{false} \text{ and } -fminN_F < x < 0 \\ &= \mathbf{underflow}(0) && \text{if } denorm_F = \mathbf{false} \text{ and } 0 < x < fminN_F \\ &= \mathbf{underflow}(-0) && \text{if } denorm_F = \mathbf{true} \text{ and } x < 0 \text{ and } rnd(x) = 0 \end{aligned}$$

$$\begin{aligned}
 &= \mathbf{inexact}(\mathit{rnd}(x)) \text{ or } \mathbf{underflow}(\mathit{rnd}(x)) \\
 &\quad \text{if } \mathit{denorm}_F = \mathbf{true} \text{ and } x \neq \mathit{rnd}(x) \text{ and } |x| < \mathit{fmin}N_F \\
 &\quad \text{and } \mathit{rnd} \text{ has no denormalisation loss at } x \\
 &= \mathbf{underflow}(\mathit{rnd}(x)) \\
 &\quad \text{otherwise}
 \end{aligned}$$

NOTES

- 2 Overflow may be detected before or after rounding. If overflow is detected before rounding, the bounds for overflow are independent of rounding.
- 3 The notifications **underflow** and **overflow** implies inexact result. When **inexact** notifications are supported, and when recording notifications in indicators, recording an **underflow** or an **overflow** notification implies also recording an **inexact** notification.
- 4 An implementation is allowed to choose between **inexact**($\mathit{rnd}(x)$) and **underflow**($\mathit{rnd}(x)$) for values of x in the interval $]-\mathit{fmin}N_F, \mathit{fmin}N_F[$ where x is not in F and there is no denormalisation loss at x . However, a subnormal value without an underflow notification can be chosen only if denorm_F is **true** and no denormalisation loss occurs at x .
- 5 In the non-conforming case $\mathit{denorm}_F = \mathbf{false}$, neither rounding is heeded in case of **underflow**.

An implementation shall document how the choice between **inexact**($\mathit{rnd}(x)$) and **underflow**($\mathit{rnd}(x)$) is made. Different floating point types may have different versions of result_F .

Define the $\mathit{no_result}_{F \rightarrow F'}$ and $\mathit{no_result2}_{F \rightarrow F'}$ helper functions:

$$\mathit{no_result}_{F \rightarrow F'} : F \rightarrow \{\mathbf{invalid}\}$$

$$\begin{aligned}
 \mathit{no_result}_{F \rightarrow F'}(x) \\
 &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \in F \cup \{-\infty, -0, +\infty\} \\
 &= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\
 &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN}
 \end{aligned}$$

$$\mathit{no_result2}_{F \rightarrow F'} : F \times F \rightarrow \{\mathbf{invalid}\}$$

$$\begin{aligned}
 \mathit{no_result2}_{F \rightarrow F'}(x, y) \\
 &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x, y \in F \cup \{-\infty, -0, +\infty\} \\
 &= \mathbf{qNaN} && \text{if at least one of } x \text{ and } y \text{ is a quiet NaN and} \\
 &\quad \text{neither a signalling NaN} \\
 &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN or } y \text{ is a signalling NaN}
 \end{aligned}$$

These helper functions are used to specify both NaN argument handling and to handle non-NaN-argument cases where **invalid**(**qNaN**) is the appropriate result.

NOTE 6 – The handling of special values, if available, other than NaNs, infinities, and -0 , is left unspecified by this document.

5.2.6 Floating point operations

5.2.6.1 Comparisons

For each provided conforming floating point datatype, the following operations shall be provided.

$$\mathit{eq}_F : F \times F \rightarrow \mathbf{Boolean}$$

$eq_F(x, y)$
 = **true**
 = **false**
 = $eq_F(0, y)$
 = $eq_F(x, 0)$
 = **false**
 = **false**
 = **invalid(false)**

if $x, y \in F \cup \{-\infty, +\infty\}$ and $x = y$
 if $x, y \in F \cup \{-\infty, +\infty\}$ and $x \neq y$
 if $x = -0$ and $y \in F \cup \{-\infty, -0, +\infty\}$
 if $x \in F \cup \{-\infty, +\infty\}$ and $y = -0$
 if x is a quiet NaN and y is not a signalling NaN
 if y is a quiet NaN and x is not a signalling NaN
 if x is a signalling NaN or y is a signalling NaN

$neq_F : F \times F \rightarrow \text{Boolean}$

$neq_F(x, y)$
 = **true**
 = **false**
 = $neq_F(0, y)$
 = $neq_F(x, 0)$
 = **true**
 = **true**
 = **invalid(true)**

if $x, y \in F \cup \{-\infty, +\infty\}$ and $x \neq y$
 if $x, y \in F \cup \{-\infty, +\infty\}$ and $x = y$
 if $x = -0$ and $y \in F \cup \{-\infty, -0, +\infty\}$
 if $y = -0$ and $x \in F \cup \{-\infty, +\infty\}$
 if x is a quiet NaN and y is not a signalling NaN
 if y is a quiet NaN and x is not a signalling NaN
 if x is a signalling NaN or y is a signalling NaN

$lss_F : F \times F \rightarrow \text{Boolean}$

$lss_F(x, y)$
 = **true**
 = **false**
 = $lss_F(0, y)$
 = $lss_F(x, 0)$
 = **true**
 = **false**
 = **false**
 = **true**
 = **invalid(false)**

if $x, y \in F$ and $x < y$
 if $x, y \in F$ and $x \geq y$
 if $x = -0$ and $y \in F \cup \{-\infty, -0, +\infty\}$
 if $y = -0$ and $x \in F \cup \{-\infty, +\infty\}$
 if $x = -\infty$ and $y \in F \cup \{+\infty\}$
 if $x = +\infty$ and $y \in F \cup \{-\infty, +\infty\}$
 if $x \in F \cup \{-\infty\}$ and $y = -\infty$
 if $x \in F$ and $y = +\infty$
 if x is a NaN or y is a NaN

$leq_F : F \times F \rightarrow \text{Boolean}$

$leq_F(x, y)$
 = **true**
 = **false**
 = $leq_F(0, y)$
 = $leq_F(x, 0)$
 = **true**
 = **false**
 = **false**
 = **true**
 = **invalid(false)**

if $x, y \in F$ and $x \leq y$
 if $x, y \in F$ and $x > y$
 if $x = -0$ and $y \in F \cup \{-\infty, -0, +\infty\}$
 if $y = -0$ and $x \in F \cup \{-\infty, +\infty\}$
 if $x = -\infty$ and $y \in F \cup \{-\infty, +\infty\}$
 if $x = +\infty$ and $y \in F \cup \{-\infty\}$
 if $x \in F$ and $y = -\infty$
 if $x \in F \cup \{+\infty\}$ and $y = +\infty$
 if x is a NaN or y is a NaN

$gtr_F : F \times F \rightarrow \text{Boolean}$

$gtr_F(x, y) = lss_F(y, x)$

$geq_F : F \times F \rightarrow \text{Boolean}$

$geq_F(x, y) = leq_F(y, x)$

$isnegzero_F : F \rightarrow \text{Boolean}$

$isnegzero_F(x)$	$= \text{true}$	if $x = -0$
	$= \text{false}$	if $x \in F \cup \{-\infty, +\infty\}$
	$= \text{invalid}(\text{false})$	if x is a NaN

$istiny_F : F \rightarrow \text{Boolean}$

$istiny_F(x)$	$= \text{true}$	if $(x \in F \text{ and } x < fminN_F)$ or $x = -0$
	$= \text{false}$	if $(x \in F \text{ and } x \geq fminN_F)$ or $x \in \{-\infty, +\infty\}$
	$= \text{invalid}(\text{false})$	if x is a NaN

$isnan_F : F \rightarrow \text{Boolean}$

$isnan_F(x)$	$= \text{false}$	if $x \in F \cup \{-\infty, -0, +\infty\}$
	$= \text{true}$	if x is a quiet NaN
	$= \text{invalid}(\text{true})$	if x is a signalling NaN

$issignan_F : F \rightarrow \text{Boolean}$

$issignan_F(x)$	$= \text{false}$	if $x \in F \cup \{-\infty, -0, +\infty\}$
	$= \text{false}$	if x is a quiet NaN
	$= \text{true}$	if x is a signalling NaN

5.2.6.2 Basic arithmetic

For each provided conforming floating point datatype, the following round to nearest operations shall be provided, and the round towards negative and positive infinity operations should be provided. For the non-conforming case that $denorm_F = \text{false}$, the operations below using up_F or $down_F$ as rounding shall *not* be provided.

NOTE 1 – If $denorm_F = \text{false}$, then any result that is smaller than $fminN_F$ is replaced by zero. This implies that neither rounding direction (nearest, up, down) is heeded, doing “flush to zero” for would-be subnormal results. Thus if $denorm_F = \text{false}$, the directed rounding operations would be unreliable for interval arithmetic, as well as other uses. That is why the directed rounding operations are not to be provided when $denorm_F = \text{false}$.

The operations in this clause are specified only for the case that $r_F = r_{F'}$, $denorm_F = denorm_{F'}$, $iec_60559_F = iec_60559_{F'}$. If $iec_60559_F = \text{false}$ then the operations are required only if $F = F'$. The $add_{F \rightarrow F'}$ and $sub_{F \rightarrow F'}$ operations can underflow only if $denorm_{F'} = \text{false}$ (non-conforming case) or $emin_F - p_F < emin_{F'} - p_{F'}$.

$neg_F : F \rightarrow F \cup \{-0\}$

$neg_F(x)$	$= -x$	if $x \in F$ and $x \neq 0$
	$= -0$	if $x = 0$
	$= 0$	if $x = -0$
	$= -\infty$	if $x = +\infty$
	$= +\infty$	if $x = -\infty$
	$= no_result_{F \rightarrow F}(x)$	otherwise

$add_{F \rightarrow F'} : F \times F \rightarrow F' \cup \{\text{inexact}, \text{underflow}, \text{overflow}\}$

$$\begin{aligned}
add_{F \rightarrow F'}(x, y) &= result_{F'}(x + y, nearest_{F'}) \\
&\quad \text{if } x, y \in F \\
&= -0 \quad \text{if } x = -0 \text{ and } y = -0 \\
&= add_{F \rightarrow F'}(0, y) \quad \text{if } x = -0 \text{ and } y \in F \cup \{-\infty, +\infty\} \\
&= add_{F \rightarrow F'}(x, 0) \quad \text{if } x \in F \cup \{-\infty, +\infty\} \text{ and } y = -0 \\
&= +\infty \quad \text{if } x = +\infty \text{ and } y \in F \cup \{+\infty\} \\
&= +\infty \quad \text{if } x \in F \text{ and } y = +\infty \\
&= -\infty \quad \text{if } x = -\infty \text{ and } y \in F \cup \{-\infty\} \\
&= -\infty \quad \text{if } x \in F \text{ and } y = -\infty \\
&= no_result2_{F \rightarrow F'}(x, y) \quad \text{otherwise}
\end{aligned}$$

$$add_{F \rightarrow F'}^{\uparrow} : F \times F \rightarrow F' \cup \{\text{inexact, underflow, overflow}\}$$

$$\begin{aligned}
add_{F \rightarrow F'}^{\uparrow}(x, y) &= result_{F'}(x + y, up_{F'}) \quad \text{if } x, y \in F \\
&= add_{F \rightarrow F'}(x, y) \quad \text{otherwise}
\end{aligned}$$

$$add_{F \rightarrow F'}^{\downarrow} : F \times F \rightarrow F' \cup \{-0, \text{inexact, underflow, overflow}\}$$

$$\begin{aligned}
add_{F \rightarrow F'}^{\downarrow}(x, y) &= result_{F'}(x + y, down_{F'}) \quad \text{if } x, y \in F \text{ and } (x + y \neq 0 \text{ or } x = 0) \\
&= -0 \quad \text{if } x, y \in F \text{ and } x + y = 0 \text{ and } x \neq 0 \\
&= -0 \quad \text{if } add_{F \rightarrow F'}(x, y) = 0 \text{ and } (x = -0 \text{ or } y = -0) \\
&= add_{F \rightarrow F'}(x, y) \quad \text{otherwise}
\end{aligned}$$

$$sub_{F \rightarrow F'} : F \times F \rightarrow F' \cup \{\text{inexact, underflow, overflow}\}$$

$$sub_{F \rightarrow F'}(x, y) = add_{F \rightarrow F'}(x, neg_F(y))$$

$$sub_{F \rightarrow F'}^{\uparrow} : F \times F \rightarrow F' \cup \{\text{inexact, underflow, overflow}\}$$

$$sub_{F \rightarrow F'}^{\uparrow}(x, y) = add_{F \rightarrow F'}^{\uparrow}(x, neg_F(y))$$

$$sub_{F \rightarrow F'}^{\downarrow} : F \times F \rightarrow F' \cup \{-0, \text{inexact, underflow, overflow}\}$$

$$sub_{F \rightarrow F'}^{\downarrow}(x, y) = add_{F \rightarrow F'}^{\downarrow}(x, neg_F(y))$$

$$mul_{F \rightarrow F'} : F \times F \rightarrow F' \cup \{-0, \text{inexact, underflow, overflow}\}$$

$$\begin{aligned}
mul_{F \rightarrow F'}(x, y) &= result_{F'}(x \cdot y, nearest_{F'}) \\
&\quad \text{if } x, y \in F \text{ and } x \neq 0 \text{ and } y \neq 0 \\
&= 0 \quad \text{if } x = 0 \text{ and } y \in F \text{ and } y \geq 0 \\
&= -0 \quad \text{if } x = 0 \text{ and } ((y \in F \text{ and } y < 0) \text{ or } y = -0) \\
&= -0 \quad \text{if } x = -0 \text{ and } y \in F \text{ and } y \geq 0 \\
&= 0 \quad \text{if } x = -0 \text{ and } ((y \in F \text{ and } y < 0) \text{ or } y = -0) \\
&= 0 \quad \text{if } x \in F \text{ and } x > 0 \text{ and } y = 0 \\
&= -0 \quad \text{if } x \in F \text{ and } x < 0 \text{ and } y = 0 \\
&= -0 \quad \text{if } x \in F \text{ and } x > 0 \text{ and } y = -0 \\
&= 0 \quad \text{if } x \in F \text{ and } x < 0 \text{ and } y = -0 \\
&= +\infty \quad \text{if } x = +\infty \text{ and } ((y \in F \text{ and } y > 0) \text{ or } y = +\infty) \\
&= -\infty \quad \text{if } x = +\infty \text{ and } ((y \in F \text{ and } y < 0) \text{ or } y = -\infty)
\end{aligned}$$

$$\begin{aligned}
 &= -\infty && \text{if } x = -\infty \text{ and } ((y \in F \text{ and } y > 0) \text{ or } y = +\infty) \\
 &= +\infty && \text{if } x = -\infty \text{ and } ((y \in F \text{ and } y < 0) \text{ or } y = -\infty) \\
 &= +\infty && \text{if } x \in F \text{ and } x > 0 \text{ and } y = +\infty \\
 &= -\infty && \text{if } x \in F \text{ and } x < 0 \text{ and } y = +\infty \\
 &= -\infty && \text{if } x \in F \text{ and } x > 0 \text{ and } y = -\infty \\
 &= +\infty && \text{if } x \in F \text{ and } x < 0 \text{ and } y = -\infty \\
 &= no_result2_{F \rightarrow F'}(x, y) && \text{otherwise}
 \end{aligned}$$

$$mul_{F \rightarrow F'}^{\uparrow} : F \times F \rightarrow F' \cup \{-0, \text{inexact}, \text{underflow}, \text{overflow}\}$$

$$\begin{aligned}
 mul_{F \rightarrow F'}^{\uparrow}(x, y) &= result_{F'}(x \cdot y, up_{F'}) && \text{if } x, y \in F \text{ and } x \neq 0 \text{ and } y \neq 0 \\
 &= mul_{F \rightarrow F'}(x, y) && \text{otherwise}
 \end{aligned}$$

$$mul_{F \rightarrow F'}^{\downarrow} : F \times F \rightarrow F' \cup \{-0, \text{inexact}, \text{underflow}, \text{overflow}\}$$

$$\begin{aligned}
 mul_{F \rightarrow F'}^{\downarrow}(x, y) &= result_{F'}(x \cdot y, down_{F'}) && \text{if } x, y \in F \text{ and } x \neq 0 \text{ and } y \neq 0 \\
 &= mul_{F \rightarrow F'}(x, y) && \text{otherwise}
 \end{aligned}$$

$$div_{F \rightarrow F'} : F \times F \rightarrow F' \cup \{-0, \text{inexact}, \text{underflow}, \text{overflow}, \text{infinitary}, \text{invalid}\}$$

$$\begin{aligned}
 div_{F \rightarrow F'}(x, y) &= result_{F'}(x/y, nearest_{F'}) \\
 & && \text{if } x, y \in F \text{ and } x \neq 0 \text{ and } y \neq 0 \\
 &= 0 && \text{if } x = 0 \text{ and } y \in F \text{ and } y > 0 \\
 &= -0 && \text{if } x = 0 \text{ and } y \in F \text{ and } y < 0 \\
 &= -0 && \text{if } x = -0 \text{ and } y \in F \text{ and } y > 0 \\
 &= 0 && \text{if } x = -0 \text{ and } y \in F \text{ and } y < 0 \\
 &= \text{infinitary}(+\infty) && \text{if } x \in F \text{ and } x > 0 \text{ and } y = 0 \\
 &= \text{infinitary}(-\infty) && \text{if } x \in F \text{ and } x < 0 \text{ and } y = 0 \\
 &= \text{infinitary}(-\infty) && \text{if } x \in F \text{ and } x > 0 \text{ and } y = -0 \\
 &= \text{infinitary}(+\infty) && \text{if } x \in F \text{ and } x < 0 \text{ and } y = -0 \\
 &= 0 && \text{if } x \in F \text{ and } x \geq 0 \text{ and } y = +\infty \\
 &= -0 && \text{if } x \in F \text{ and } x \geq 0 \text{ and } y = -\infty \\
 &= -0 && \text{if } ((x \in F \text{ and } x < 0) \text{ or } x = -0) \text{ and } y = +\infty \\
 &= 0 && \text{if } ((x \in F \text{ and } x < 0) \text{ or } x = -0) \text{ and } y = -\infty \\
 &= +\infty && \text{if } x = +\infty \text{ and } y \in F \text{ and } y \geq 0 \\
 &= -\infty && \text{if } x = -\infty \text{ and } y \in F \text{ and } y \geq 0 \\
 &= -\infty && \text{if } x = +\infty \text{ and } ((y \in F \text{ and } y < 0) \text{ or } y = -0) \\
 &= +\infty && \text{if } x = -\infty \text{ and } ((y \in F \text{ and } y < 0) \text{ or } y = -0) \\
 &= no_result2_{F \rightarrow F'}(x, y) && \text{otherwise}
 \end{aligned}$$

$$div_{F \rightarrow F'}^{\uparrow} : F \times F \rightarrow F' \cup \{-0, \text{inexact}, \text{underflow}, \text{overflow}, \text{infinitary}, \text{invalid}\}$$

$$\begin{aligned}
 div_{F \rightarrow F'}^{\uparrow}(x, y) &= result_{F'}(x/y, up_{F'}) && \text{if } x, y \in F \text{ and } x \neq 0 \text{ and } y \neq 0 \\
 &= div_{F \rightarrow F'}(x, y) && \text{otherwise}
 \end{aligned}$$

$$div_{F \rightarrow F'}^{\downarrow} : F \times F \rightarrow F' \cup \{-0, \text{inexact}, \text{underflow}, \text{overflow}, \text{infinitary}, \text{invalid}\}$$

$$\begin{aligned}
 div_{F \rightarrow F'}^{\downarrow}(x, y) &= result_{F'}(x/y, down_{F'}) && \text{if } x, y \in F \text{ and } x \neq 0 \text{ and } y \neq 0 \\
 &= div_{F \rightarrow F'}(x, y) && \text{otherwise}
 \end{aligned}$$

$$abs_F : F \rightarrow F$$

$$\begin{aligned} abs_F(x) &= |x| && \text{if } x \in F \\ &= 0 && \text{if } x = -0 \\ &= +\infty && \text{if } x \in \{-\infty, +\infty\} \\ &= no_result_{F \rightarrow F}(x) && \text{otherwise} \end{aligned}$$

$$signum_F : F \rightarrow F$$

$$\begin{aligned} signum_F(x) &= 1 && \text{if } (x \in F \text{ and } x \geq 0) \text{ or } x = +\infty \\ &= -1 && \text{if } (x \in F \text{ and } x < 0) \text{ or } x \in \{-0, -\infty\} \\ &= no_result_{F \rightarrow F}(x) && \text{otherwise} \end{aligned}$$

$$residue_F : F \times F \rightarrow F \cup \{-0, \text{invalid}\}$$

$$\begin{aligned} residue_F(x, y) &= result_F(x - (\text{round}(x/y) \cdot y), nearest_F) && \text{if } x, y \in F \text{ and } y \neq 0 \text{ and} \\ & && (x \geq 0 \text{ or } x - (\text{round}(x/y) \cdot y) \neq 0) \\ &= -0 && \text{if } x, y \in F \text{ and } y \neq 0 \text{ and} \\ & && x < 0 \text{ and } x - (\text{round}(x/y) \cdot y) = 0 \\ &= -0 && \text{if } x = -0 \text{ and } y \in F \cup \{-\infty, +\infty\} \text{ and } y \neq 0 \\ &= x && \text{if } x \in F \text{ and } y \in \{-\infty, +\infty\} \\ &= no_result_{2F \rightarrow F}(x, y) && \text{otherwise} \end{aligned}$$

NOTE 2 – The $residue_F$ operation is informally known as “IEEE remainder”.

$$sqr_{F \rightarrow F'} : F \rightarrow F' \cup \{\text{inexact}, \text{underflow}, \text{invalid}\}$$

$$\begin{aligned} sqr_{F \rightarrow F'}(x) &= result_{F'}(\sqrt{x}, nearest_{F'}) \\ & && \text{if } x \in F \text{ and } x \geq 0 \\ &= x && \text{if } x \in \{-0, +\infty\} \\ &= no_result_{F \rightarrow F'}(x) && \text{otherwise} \end{aligned}$$

$$sqr_{F \rightarrow F'}^{\uparrow} : F \rightarrow F' \cup \{\text{inexact}, \text{underflow}, \text{invalid}\}$$

$$\begin{aligned} sqr_{F \rightarrow F'}^{\uparrow}(x, y) &= result_{F'}(\sqrt{x}, up_{F'}) && \text{if } x \in F \text{ and } x \geq 0 \\ &= sqr_{F \rightarrow F'}(x) && \text{otherwise} \end{aligned}$$

$$sqr_{F \rightarrow F'}^{\downarrow} : F \rightarrow F' \cup \{\text{inexact}, \text{underflow}, \text{invalid}\}$$

$$\begin{aligned} sqr_{F \rightarrow F'}^{\downarrow}(x, y) &= result_{F'}(\sqrt{x}, down_{F'}) && \text{if } x \in F \text{ and } x \geq 0 \\ &= sqr_{F \rightarrow F'}(x) && \text{otherwise} \end{aligned}$$

5.2.6.3 Value dissection

For each provided floating point type, the following operations shall be provided. For the non-conforming case of $denorm_F = \text{false}$, ulp_F may underflow, and the operations $succ_F$ and $pred_F$ shall *not* be provided.

5.2.6 Floating point operations

27

The $exponent_{F \rightarrow I}$ and $scale_{F,I}$ operations are specified for an integer datatype I where $minint_I < emin_F - p_F$ and $maxint_I > emax_F$.

$$\begin{aligned}
 exponent_{F \rightarrow I} : F &\rightarrow I \cup \{\mathbf{infinitary}\} \\
 exponent_{F \rightarrow I}(x) &= \lfloor \log_{r_F}(|x|) \rfloor + 1 && \text{if } x \in F \text{ and } x \neq 0 \\
 &= \mathbf{infinitary}(-\infty) && \text{if } x \in \{-0, 0\} \\
 &= +\infty && \text{if } x \in \{-\infty, +\infty\} \\
 &= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\
 &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN}
 \end{aligned}$$

NOTES

- 1 Since most integer datatypes cannot represent infinitary or NaN values, documented out of range finite integer values of the correct sign may be used instead of the infinities here.
- 2 The related IEC 60559 operation *logb* returns a floating point value, to guarantee the representability of the infinitary (and NaN) return values.

$$\begin{aligned}
 fraction_F : F &\rightarrow F \\
 fraction_F(x) &= x / r_F^{exponent_{F \rightarrow I}(x)} && \text{if } x \in F \text{ and } x \neq 0 \\
 &= x && \text{if } x \in \{-\infty, -0, 0, +\infty\} \\
 &= no_result_{F \rightarrow F}(x) && \text{otherwise}
 \end{aligned}$$

$$\begin{aligned}
 scale_{F,I} : F \times I &\rightarrow F \cup \{\mathbf{underflow}, \mathbf{overflow}\} \\
 scale_{F,I}(x, n) &= result_F(x \cdot r_F^n, nearest_F) && \text{if } x \in F \text{ and } n \in I \\
 &= mul_{F \rightarrow F}(x, 0) && \text{if } n = -\infty \\
 &= x && \text{if } n = -0 \\
 &= mul_{F \rightarrow F}(x, convert_{I \rightarrow F}(n)) && \text{otherwise}
 \end{aligned}$$

$$\begin{aligned}
 succ_F : F &\rightarrow F \cup \{\mathbf{overflow}\} \\
 succ_F(x) &= result_F(\min \{z \in F^\dagger \mid z > x\}, nearest_F) && \text{if } x \in F \text{ and } x \neq -fmin_F \text{ and } x \neq 0 \\
 &= -fmax_F && \text{if } x = -\infty \\
 &= -0 && \text{if } x = -fmin_F \\
 &= succ_F(0) && \text{if } x = -0 \\
 &= fmin_F && \text{if } x = 0 \\
 &= +\infty && \text{if } x = +\infty \\
 &= no_result_{F \rightarrow F}(x) && \text{otherwise}
 \end{aligned}$$

$$\begin{aligned}
 pred_F : F &\rightarrow F \cup \{\mathbf{overflow}\} \\
 pred_F(x) &= neg_F(succ_F(neg_F(x)))
 \end{aligned}$$

$$\begin{aligned}
 ulp_F : F &\rightarrow F \cup \{\mathbf{underflow}\} \\
 ulp_F(x) &= result_F(u_F(x), nearest_F) && \text{if } x \in F \\
 &= ulp_F(0) && \text{if } x = -0 \\
 &= no_result_{F \rightarrow F}(x) && \text{otherwise}
 \end{aligned}$$

5.2.6.4 Value splitting

For each provided floating point type, the following operations shall be provided. The $trunc_{F,I}$ and $round_{F,I}$ operations are specified for an integer type I where $maxint_I > p_F$.

$$intpart_F : F \rightarrow F \cup \{-0\}$$

$$\begin{aligned} intpart_F(x) &= \lfloor x \rfloor && \text{if } x \in F \text{ and } x \geq 0 \\ &= neg_F(intpart_F(-x)) && \text{if } x \in F \text{ and } x < 0 \\ &= x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &= no_result_{F \rightarrow F}(x) && \text{otherwise} \end{aligned}$$

$$fractpart_F : F \rightarrow F \cup \{-0\}$$

$$\begin{aligned} fractpart_F(x) &= x - \lfloor x \rfloor && \text{if } x \in F \text{ and } x \geq 0 \\ &= neg_F(fractpart_F(-x)) && \text{if } x \in F \text{ and } x < 0 \\ &= x && \text{if } x = -0 \\ &= no_result_{F \rightarrow F}(x) && \text{otherwise} \end{aligned}$$

$$trunc_{F,I} : F \times I \rightarrow F \cup \{-0\}$$

$$\begin{aligned} trunc_{F,I}(x, n) &= \lfloor x/r_F^{e_F(x)-n} \rfloor \cdot r_F^{e_F(x)-n} && \text{if } x \in F \text{ and } x \geq 0 \text{ and } n \in I \\ &= neg_F(trunc_{F,I}(-x, n)) && \text{if } x \in F \text{ and } x < 0 \text{ and } n \in I \\ &= x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &= no_result_{F \rightarrow F}(x, n) && \text{otherwise} \end{aligned}$$

$$round_{F,I} : F \times I \rightarrow F \cup \{-0, \text{overflow}\}$$

$$\begin{aligned} round_{F,I}(x, n) &= result_F(round(x/r_F^{e_F(x)-n}) \cdot r_F^{e_F(x)-n}, nearest_F) && \text{if } x \in F \text{ and } x \geq 0 \text{ and } n \in I \\ &= neg_F(round_{F,I}(-x, n)) && \text{if } x \in F \text{ and } x < 0 \text{ and } n \in I \\ &= x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &= no_result_{F \rightarrow F}(x, n) && \text{otherwise} \end{aligned}$$

5.3 Operations for conversion between numeric datatypes

Numeric conversion between different representation forms for integer and floating point values can take place under a number of different circumstances. E.g.:

- a) explicit or implicit conversion between different numeric datatypes conforming to this part of ISO/IEC 10967;
- b) explicit or implicit conversion between different numeric datatypes only one of which conforms to this part of ISO/IEC 10967;
- c) explicit or implicit conversion between a character string and a numeric datatype.

The last case includes outputting a numeric value as a character string, inputting a numeric value from a character string source, and converting a numeral in the source program to a value in a numeric datatype (see clause 5.4). This part of ISO/IEC 10967 covers only the cases where at least one of the source and target of a conversion is a numeric datatype conforming to this part of ISO/IEC 10967.

When a character string is involved as either source or target of a conversion, this part of ISO/IEC 10967 does not specify the lexical syntax for the numerals parsed or formed. A binding standard should specify the lexical syntax or syntaxes for these numerals, and, when appropriate, how the lexical syntax for the numerals can be altered. This could include which set of digits to use in a numeral position system (Latin-Arabic digits, Arabic-Indic digits, traditional Thai digits, Devanagari digits, etc.). With the exception of the radix used in numerals expressing non-integer values, differences in lexical syntactic details that do not affect the value in $\mathcal{R}$ denoted by the numerals will not affect the result of a conforming conversion.

Output of floating point values is quite often to a fixed point format. Therefore conversion from a floating point datatype to a fixed point datatype is included in this clause.

Character string representations for integer values can include representations for -0 , $+\infty$, $-\infty$, and quiet NaNs. Character string representations for floating point and fixed point values should have formats for -0 , $+\infty$, $-\infty$, and quiet NaNs. For both integer and floating point values, character strings that are not numerals nor special values according to the lexical syntax used, shall be regarded as signalling NaNs when used as source of a numerical conversion.

For the cases where one of the datatypes involved in the conversion does not conform to this part of ISO/IEC 10967, the values of some numeric datatype parameters need to be inferred. For integers, one need to infer the value for *bounded*, and if that is **true** then also values for *maxint* and *minint*, and for string formats also the *radix*. For floating point values, one need to infer the values for *r*, *p*, and *emax* or *emin*. In case a precise determination is not possible, values that are ‘safe’ for that instance should be used. ‘Safe’ values for otherwise undetermined inferred parameters are such that

- a) if the value resulting from the conversion is converted back to the source datatype by a conversion conforming to this part of ISO/IEC 10967 the original value, or a close approximation, should be regenerated if possible, and
- b) overflow and underflow are avoided if possible.

If, and only if, a specified infinite special value result cannot be represented in the target datatype, the infinity result shall be interpreted as implying the **infinitary** notification. If, and only if, a specified NaN special value result cannot be represented in the target datatype, the NaN result shall be interpreted as implying the **invalid** notification. If, and only if, a specified -0 special value result cannot be represented in the target datatype, the -0 result shall be interpreted as 0.

5.3.1 Integer to integer conversions

Let I and I' be non-special value sets for integer datatypes. At least one of the datatypes corresponding to I and I' conforms to this part of ISO/IEC 10967.

The $convert_{I \rightarrow I'}$ operation:

$$\begin{aligned}
 &convert_{I \rightarrow I'} : I \rightarrow I' \cup \{\mathbf{overflow}\} \\
 &convert_{I \rightarrow I'}(x) = \begin{aligned} &result_{I'}(x) && \text{if } x \in I \\ &x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &\mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\ &\mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN} \end{aligned}
 \end{aligned}$$

5.3.2 Floating point to integer conversions

Let F be the non-special value set for a floating point datatype. Let I be the non-special value set for an integer datatype. At least one of the datatypes corresponding to F and I conforms to this part of ISO/IEC 10967.

The $\text{floor}_{F \rightarrow I}$, $\text{rounding}_{F \rightarrow I}$, and $\text{ceiling}_{F \rightarrow I}$ operations:

$$\begin{aligned} \text{floor}_{F \rightarrow I} : F &\rightarrow I \cup \{\mathbf{overflow}\} \\ \text{floor}_{F \rightarrow I}(x) &= \text{result}_I(\lfloor x \rfloor) && \text{if } x \in F \\ &= x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\ &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN} \end{aligned}$$

$$\begin{aligned} \text{rounding}_{F \rightarrow I} : F &\rightarrow I \cup \{-0, \mathbf{overflow}\} \\ \text{rounding}_{F \rightarrow I}(x) &= \text{result}_I(\text{round}(x)) && \text{if } x \in F \text{ and } (x \geq 0 \text{ or } \text{round}(x) \neq 0) \\ &= -0 && \text{if } x \in F \text{ and } x < 0 \text{ and } \text{round}(x) = 0 \\ &= x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\ &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN} \end{aligned}$$

$$\begin{aligned} \text{ceiling}_{F \rightarrow I} : F &\rightarrow I \cup \{-0, \mathbf{overflow}\} \\ \text{ceiling}_{F \rightarrow I}(x) &= \text{result}_I(\lceil x \rceil) && \text{if } x \in F \text{ and } (x \geq 0 \text{ or } \lceil x \rceil \neq 0) \\ &= -0 && \text{if } x \in F \text{ and } x < 0 \text{ and } \lceil x \rceil = 0 \\ &= x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\ &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN} \end{aligned}$$

5.3.3 Integer to floating point conversions

Let I be the non-special value set for an integer datatype. Let F be the non-special value set for a floating point datatype. At least one of the datatypes corresponding to I and F conforms to this part of ISO/IEC 10967.

The $\text{convert}_{I \rightarrow F}$ operation:

$$\begin{aligned} \text{convert}_{I \rightarrow F} : I &\rightarrow F \cup \{\mathbf{inexact}, \mathbf{overflow}\} \\ \text{convert}_{I \rightarrow F}(x) &= \text{result}_F(x, \text{nearest}_F) && \text{if } x \in I \\ &= x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\ &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN} \end{aligned}$$

The $\text{convert}_{I \rightarrow F}^{\uparrow}$ and $\text{convert}_{I \rightarrow F}^{\downarrow}$ operations:

$$\begin{aligned} \text{convert}_{I \rightarrow F}^{\uparrow} : I &\rightarrow F \cup \{\mathbf{inexact}, \mathbf{overflow}\} \\ \text{convert}_{I \rightarrow F}^{\uparrow}(x) &= \text{result}_F(x, \text{up}_F) && \text{if } x \in I \\ &= \text{convert}_{I \rightarrow F}(x) && \text{otherwise} \end{aligned}$$

$$\begin{aligned} \text{convert}_{I \rightarrow F}^{\downarrow} : I &\rightarrow F \cup \{\mathbf{inexact}, \mathbf{overflow}\} \\ \text{convert}_{I \rightarrow F}^{\downarrow}(x) &= \text{result}_F(x, \text{down}_F) && \text{if } x \in I \\ &= \text{convert}_{I \rightarrow F}(x) && \text{otherwise} \end{aligned}$$

5.3.4 Floating point to floating point conversions

Let F and F' be non-special value sets for floating point datatypes. At least one of the datatypes corresponding to F and F' conforms to this part of ISO/IEC 10967.

The $\text{convert}_{F \rightarrow F'}$ operation:

$$\begin{aligned} \text{convert}_{F \rightarrow F'} : F &\rightarrow F' \cup \{\mathbf{inexact}, \mathbf{underflow}, \mathbf{overflow}\} \\ \text{convert}_{F \rightarrow F'}(x) &= \text{result}_{F'}(x, \text{nearest}_{F'}) && \text{if } x \in F \\ &= x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\ &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN} \end{aligned}$$

NOTE – Modern techniques allow, on the average, efficient conversion with a maximum error of 0.5 ulp even when the radices differ. C99 [15], for instance, requires that all floating point value conversion is done with a maximum error of 0.5 ulp.

The $\text{convert}_{F \rightarrow F'}^{\uparrow}$ and $\text{convert}_{F \rightarrow F'}^{\downarrow}$ operations:

$$\begin{aligned} \text{convert}_{F \rightarrow F'}^{\uparrow} : F &\rightarrow F' \cup \{\mathbf{inexact}, \mathbf{underflow}, \mathbf{overflow}\} \\ \text{convert}_{F \rightarrow F'}^{\uparrow}(x) &= \text{result}_{F'}(x, \text{up}_{F'}) && \text{if } x \in F \\ &= \text{convert}_{F \rightarrow F'} && \text{otherwise} \\ \\ \text{convert}_{F \rightarrow F'}^{\downarrow} : F &\rightarrow F' \cup \{\mathbf{inexact}, \mathbf{underflow}, \mathbf{overflow}\} \\ \text{convert}_{F \rightarrow F'}^{\downarrow}(x) &= \text{result}_{F'}(x, \text{down}_{F'}) && \text{if } x \in F \\ &= \text{convert}_{F \rightarrow F'} && \text{otherwise} \end{aligned}$$

5.3.5 Floating point to fixed point conversions

Let F be the non-special value set for a floating point datatype conforming to this part of ISO/IEC 10967. Let D be the non-special value set for a fixed point datatype.

A fixed point datatype D is a subset of $\mathcal{R}$, characterised by a radix, $r_D \in \mathcal{Z}$ (≥ 2), a density, $d_D \in \mathcal{Z}$ (≥ 0), and if it is bounded, a maximum positive value, $\text{dmax}_D \geq 1$. Given these values, the following sets are defined:

$$D^* = \{n/(r_D^{d_D}) \mid n \in \mathcal{Z}\}$$

$$\begin{aligned} D &= D^* && \text{if } D \text{ is not bounded} \\ D &= D^* \cap [-\text{dmax}_D, \text{dmax}_D] && \text{if } D \text{ is bounded} \end{aligned}$$

NOTE 1 – D corresponds to **scaled**(r_D , d_D) in ISO/IEC 11404 *Information technology – General-Purpose Datatypes (GPD)* [10]. However, that standard does not specify a dmax_D parameter.

The fixed point rounding helper functions:

$$\text{nearest}_D : \mathcal{R} \rightarrow D^*$$

is the rounding function that rounds to nearest, ties round to even last digit.

$$up_D : \mathcal{R} \rightarrow D^*$$

is the rounding function that rounds towards positive infinity.

$$down_D : \mathcal{R} \rightarrow D^*$$

is the rounding function that rounds towards negative infinity.

The fixed point result helper function, $result_D$, is like $result_F$, but for a fixed point datatype. It will return **overflow** if the rounded result is not representable:

$$\begin{aligned} result_D : \mathcal{R} \times (\mathcal{R} \rightarrow D^*) &\rightarrow D \cup \{\mathbf{inexact}, \mathbf{overflow}\} \\ result_D(x, rnd) &= \mathbf{overflow}(+\infty) && \text{if } x \in \mathcal{R} \text{ and } rnd(x) \notin D \text{ and } x > 0 \\ &= \mathbf{overflow}(-\infty) && \text{if } x \in \mathcal{R} \text{ and } rnd(x) \notin D \text{ and } x < 0 \\ &= x && \text{if } rnd(x) \in D \text{ and } rnd(x) = x \\ &= \mathbf{inexact}(rnd(x)) && \text{if } rnd(x) \in D \text{ and } rnd(x) \neq x \text{ and} \\ &&& (rnd(x) \neq 0 \text{ or } 0 < x) \\ &= \mathbf{inexact}(-0) && \text{if } rnd(x) = 0 \text{ and } x < 0 \end{aligned}$$

The $convert_{F \rightarrow D}$ operation:

$$\begin{aligned} convert_{F \rightarrow D} : F &\rightarrow D \cup \{\mathbf{inexact}, \mathbf{overflow}\} \\ convert_{F \rightarrow D}(x) &= result_D(x, nearest_D) && \text{if } x \in F \\ &= x && \text{if } x \in \{-\infty, -0, +\infty\} \\ &= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\ &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN} \end{aligned}$$

NOTES

- 2 The datatype D need not be visible in the programming language. D may be a subtype of strings, according to some format. Even so, no datatype for strings need be present in the programming language.
- 3 This covers, among other things, “output” of floating point datatype values, to fixed point string formats. E.g. a binding may say that `float_to_fixed_string( $x$ ,  $m$ ,  $n$ )` is bound to $convert_{F \rightarrow S_{m,n}}(x)$ where $S_{m,n}$ is strings of length m , representing fixed point values in radix 10 with n decimals. The binding should also detail how NaNs, signed zeroes and infinities are represented in $S_{m,n}$, as well as the precise format of the strings representing ordinary values. (Note that if the length of the target string is limited, the conversion may overflow.)

The $convert_{F \rightarrow D}^\uparrow$ and $convert_{F \rightarrow D}^\downarrow$ operations:

$$\begin{aligned} convert_{F \rightarrow D}^\uparrow : F &\rightarrow D \cup \{\mathbf{inexact}, \mathbf{overflow}\} \\ convert_{F \rightarrow D}^\uparrow(x) &= result_D(x, up_D) && \text{if } x \in F \\ &= convert_{F \rightarrow D}(x) && \text{otherwise} \end{aligned}$$

$$\begin{aligned} convert_{F \rightarrow D}^\downarrow : F &\rightarrow D \cup \{\mathbf{inexact}, \mathbf{overflow}\} \\ convert_{F \rightarrow D}^\downarrow(x) &= result_D(x, down_D) && \text{if } x \in F \\ &= convert_{F \rightarrow D}(x) && \text{otherwise} \end{aligned}$$