
**Information technology — Language
independent arithmetic —**

**Part 2:
Elementary numerical functions**

*Technologies de l'information — Arithmétique de langage indépendant —
Partie 2: Fonctions numériques élémentaires*

Reference number
ISO/IEC 10967-2:2001(E)



PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2001

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.ch
Web www.iso.ch

Printed in Switzerland

Contents

Foreword	viii
Introduction	ix
1 Scope	1
1.1 Inclusions	1
1.2 Exclusions	2
2 Conformity	2
3 Normative references	3
4 Symbols and definitions	4
4.1 Symbols	4
4.1.1 Sets and intervals	4
4.1.2 Operators and relations	4
4.1.3 Mathematical functions	5
4.1.4 Exceptional values	5
4.1.5 Datatypes	6
4.2 Definitions of terms	7
5 Specifications for integer and floating point operations	10
5.1 Basic integer operations	10
5.1.1 The integer <i>result</i> and <i>wrap</i> helper functions	10
5.1.2 Integer maximum and minimum	11
5.1.3 Integer diminish	11
5.1.4 Integer power and arithmetic shift	12
5.1.5 Integer square root	12
5.1.6 Divisibility tests	12
5.1.7 Integer division (with floor, round, or ceiling) and remainder	13
5.1.8 Greatest common divisor and least common positive multiple	13
5.1.9 Support operations for extended integer range	14
5.2 Basic floating point operations	15
5.2.1 The rounding and floating point <i>result</i> helper functions	15
5.2.2 Floating point maximum and minimum	17
5.2.3 Floating point diminish	18
5.2.4 Floor, round, and ceiling	19
5.2.5 Remainder after division with round to integer	20
5.2.6 Square root and reciprocal square root	20
5.2.7 Multiplication to higher precision floating point datatype	20
5.2.8 Support operations for extended floating point precision	21
5.3 Elementary transcendental floating point operations	22
5.3.1 Maximum error requirements	22
5.3.2 Sign requirements	23
5.3.3 Monotonicity requirements	23
5.3.4 The <i>result*</i> helper function	23
5.3.5 Hypotenuse	24
5.3.6 Operations for exponentiations and logarithms	24

5.3.6.1	Integer power of argument base	24
5.3.6.2	Natural exponentiation	25
5.3.6.3	Natural exponentiation, minus one	26
5.3.6.4	Exponentiation of 2	27
5.3.6.5	Exponentiation of 10	27
5.3.6.6	Exponentiation of argument base	28
5.3.6.7	Exponentiation of one plus the argument base, minus one	29
5.3.6.8	Natural logarithm	29
5.3.6.9	Natural logarithm of one plus the argument	30
5.3.6.10	2-logarithm	30
5.3.6.11	10-logarithm	31
5.3.6.12	Argument base logarithm	31
5.3.6.13	Argument base logarithm of one plus each argument	32
5.3.7	Introduction to operations for trigonometric elementary functions	32
5.3.8	Operations for radian trigonometric elementary functions	33
5.3.8.1	Radian angle normalisation	34
5.3.8.2	Radian sine	35
5.3.8.3	Radian cosine	35
5.3.8.4	Radian tangent	36
5.3.8.5	Radian cotangent	36
5.3.8.6	Radian secant	37
5.3.8.7	Radian cosecant	37
5.3.8.8	Radian cosine with sine	38
5.3.8.9	Radian arc sine	38
5.3.8.10	Radian arc cosine	38
5.3.8.11	Radian arc tangent	39
5.3.8.12	Radian arc cotangent	40
5.3.8.13	Radian arc secant	41
5.3.8.14	Radian arc cosecant	41
5.3.8.15	Radian angle from Cartesian co-ordinates	42
5.3.9	Operations for trigonometrics with given angular unit	43
5.3.9.1	Argument angular-unit angle normalisation	43
5.3.9.2	Argument angular-unit sine	44
5.3.9.3	Argument angular-unit cosine	45
5.3.9.4	Argument angular-unit tangent	45
5.3.9.5	Argument angular-unit cotangent	46
5.3.9.6	Argument angular-unit secant	47
5.3.9.7	Argument angular-unit cosecant	47
5.3.9.8	Argument angular-unit cosine with sine	48
5.3.9.9	Argument angular-unit arc sine	48
5.3.9.10	Argument angular-unit arc cosine	48
5.3.9.11	Argument angular-unit arc tangent	49
5.3.9.12	Argument angular-unit arc cotangent	50
5.3.9.13	Argument angular-unit arc secant	51
5.3.9.14	Argument angular-unit arc cosecant	51
5.3.9.15	Argument angular-unit angle from Cartesian co-ordinates	52
5.3.10	Operations for angular-unit conversions	53
5.3.10.1	Converting radian angle to argument angular-unit angle	53

5.3.10.2	Converting argument angular-unit angle to radian angle	54
5.3.10.3	Converting argument angular-unit angle to (another) argument angular-unit angle	55
5.3.11	Operations for hyperbolic elementary functions	56
5.3.11.1	Hyperbolic sine	56
5.3.11.2	Hyperbolic cosine	56
5.3.11.3	Hyperbolic tangent	57
5.3.11.4	Hyperbolic cotangent	58
5.3.11.5	Hyperbolic secant	58
5.3.11.6	Hyperbolic cosecant	59
5.3.11.7	Inverse hyperbolic sine	59
5.3.11.8	Inverse hyperbolic cosine	60
5.3.11.9	Inverse hyperbolic tangent	60
5.3.11.10	Inverse hyperbolic cotangent	60
5.3.11.11	Inverse hyperbolic secant	61
5.3.11.12	Inverse hyperbolic cosecant	61
5.4	Operations for conversion between numeric datatypes	62
5.4.1	Integer to integer conversions	63
5.4.2	Floating point to integer conversions	63
5.4.3	Integer to floating point conversions	64
5.4.4	Floating point to floating point conversions	64
5.4.5	Floating point to fixed point conversions	65
5.4.6	Fixed point to floating point conversions	66
5.5	Numerals as operations in a programming language	67
5.5.1	Numerals for integer datatypes	67
5.5.2	Numerals for floating point datatypes	68
6	Notification	68
6.1	Continuation values	69
7	Relationship with language standards	69
8	Documentation requirements	70
Annex A	(normative) Partial conformity	73
A.1	Maximum error relaxation	73
A.2	Extra accuracy requirements relaxation	74
A.3	Relationships to other operations relaxation	74
A.4	Very-close-to-axis angular normalisation relaxation	74
A.5	Part 1 requirements relaxation	75
Annex B	(informative) Rationale	77
B.1	Scope	77
B.1.1	Inclusions	77
B.1.2	Exclusions	78
B.2	Conformity	78
B.2.1	Validation	79
B.3	Normative references	79
B.4	Symbols and definitions	79

B.4.1	Symbols	79
B.4.1.1	Sets and intervals	79
B.4.1.2	Operators and relations	80
B.4.1.3	Mathematical functions	80
B.4.1.4	Exceptional values	80
B.4.1.5	Datatypes	81
B.4.2	Definitions of terms	81
B.5	Specifications for the numerical functions	81
B.5.1	Basic integer operations	82
B.5.1.1	The integer <i>result</i> and <i>wrap</i> helper functions	82
B.5.1.2	Integer maximum and minimum	82
B.5.1.3	Integer diminish	82
B.5.1.4	Integer power and arithmetic shift	83
B.5.1.5	Integer square root	83
B.5.1.6	Divisibility tests	83
B.5.1.7	Integer division (with floor, round, or ceiling) and remainder . . .	83
B.5.1.8	Greatest common divisor and least common positive multiple . . .	84
B.5.1.9	Support operations for extended integer range	84
B.5.2	Basic floating point operations	84
B.5.2.1	The rounding and floating point <i>result</i> helper functions	86
B.5.2.2	Floating point maximum and minimum	86
B.5.2.3	Floating point diminish	86
B.5.2.4	Floor, round, and ceiling	86
B.5.2.5	Remainder after division and round to integer	87
B.5.2.6	Square root and reciprocal square root	87
B.5.2.7	Multiplication to higher precision floating point datatype	88
B.5.2.8	Support operations for extended floating point precision	88
B.5.3	Elementary transcendental floating point operations	89
B.5.3.1	Maximum error requirements	89
B.5.3.2	Sign requirements	90
B.5.3.3	Monotonicity requirements	90
B.5.3.4	The <i>result*</i> helper function	90
B.5.3.5	Hypotenuse	91
B.5.3.6	Operations for exponentiations and logarithms	91
B.5.3.7	Introduction to operations for trigonometric elementary functions	93
B.5.3.8	Operations for radian trigonometric elementary functions	94
B.5.3.9	Operations for trigonometrics with given angular unit	96
B.5.3.10	Operations for angular-unit conversions	97
B.5.3.11	Operations for hyperbolic elementary functions	98
B.5.4	Operations for conversion between numeric datatypes	98
B.5.5	Numerals as operations in a programming language	99
B.5.5.1	Numerals for integer datatypes	99
B.5.5.2	Numerals for floating point datatypes	99
B.6	Notification	100
B.6.1	Continuation values	100
B.7	Relationship with language standards	101
B.8	Documentation requirements	101

Annex C (informative) Example bindings for specific languages	103
C.1 Ada	104
C.2 BASIC	110
C.3 C	114
C.4 C++	120
C.5 Fortran	126
C.6 Haskell	132
C.7 Java	137
C.8 Common Lisp	142
C.9 ISLisp	147
C.10 Modula-2	152
C.11 Pascal and Extended Pascal	157
C.12 PL/I	162
C.13 SML	167
 Annex D (informative) Bibliography	 173
 Annex E (informative) Possible changes to part 1	 177

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. 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 part of ISO/IEC 10967 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 10967-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

ISO/IEC 10967 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*

Additional parts will specify other arithmetic datatypes or arithmetic operations.

Annex A forms a normative part of this part of ISO/IEC 10967. Annexes B to E are for information only.

Introduction

The aims

Portability is a key issue for scientific and numerical software in today's heterogeneous computing environment. Such software may be required to run on systems ranging from personal computers to high performance pipelined vector processors and massively parallel systems, and the source code may be ported between several programming languages. Part 1 of ISO/IEC 10967 specifies the basic properties of integer and floating point types that can be relied upon in writing portable software.

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 operations by a conforming language processor.

It is the intent of this part to help to redress these shortcomings, by setting out precise definitions of elementary numerical functions, and requirements for documentation.

It is not claimed that this part 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. Rather, this International Standard will provide a firmer basis than hitherto for attempting such analysis.

The aims for this part, part 2 of ISO/IEC 10967, are extensions of the aims for part 1: to ensure adequate accuracy for numerical computation, predictability, notification on the production of exceptional results, and compatibility with programming language standards.

The content

The content of this part is based on part 1, and extends part 1's specifications to specifications for operations approximating real elementary functions, operations often required (usually without a detailed specification) by the standards for programming languages widely used for scientific software. This part also provides specifications for conversions between the "internal" values of an arithmetic datatype, and a very close approximation in, e.g., the decimal radix. It does not cover the further transformation to decimal string format, which is usually provided by language standards. This part also includes specifications for a number of other functions deemed useful, even though they may not be stipulated by programming language standards.

The numerical functions covered by this part are computer approximations to mathematical functions of one or more real arguments. Accuracy versus performance requirements often vary with the application at hand. This is recognised by recommending that implementors support more than one library of these numerical functions. Various documentation and (program available) parameters requirements are specified to assist programmers in the selection of the library best suited to the application at hand.

The benefits

Adoption and proper use of this part can lead to the following benefits.

Language standards will be able to define their arithmetic semantics more precisely without preventing the efficient implementation of their 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.

Programs 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 2: Elementary numerical functions

1 Scope

This part of ISO/IEC 10967 defines the properties of numerical approximations for many of the real elementary numerical functions available in standard libraries for a variety of programming languages in common use for mathematical and numerical applications.

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

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

1.1 Inclusions

The specifications of part 1 are included by reference in this part.

This part provides specifications for numerical functions for which all operand values are of integer or floating point datatypes satisfying the requirements of part 1. 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, is prescribed for each specified floating point operation.

This part covers most numerical functions required by the ISO/IEC standards for Ada [11], Basic [16], C [17], C++ [18], Fortran [22], ISLisp [24], Pascal [27], and PL/I [29]. In particular, specifications are provided for:

- a) Some additional integer operations.
- b) Some additional non-transcendental floating point operations, including maximum and minimum operations.
- c) Exponentiations, logarithms, and hyperbolics.
- d) Trigonometrics, both in radians and for argument-given angular unit with degrees as a special case.

This part also provides specifications for:

- e) Conversions between integer and floating point datatypes (possibly with different radices) conforming to the requirements of part 1, and the conversion operations used, for example, in text input and output of integer and floating point numbers.
- f) The results produced by an included floating point operation when one or more argument values are IEC 60559 special values.
- g) Program-visible parameters that characterise certain aspects of the operations.

1.2 Exclusions

This part provides no specifications for

- a) Numerical functions whose operands are of more than one datatype (with one exception). This part neither requires nor excludes the presence of such “mixed operand” operations.
- b) An interval datatype, or the operations on such data. This part neither requires nor excludes such data or operations.
- c) A fixed point datatype, or the operations on such data. This part neither requires nor excludes such data or operations.
- d) A rational datatype, or the operations on such data. This part neither requires nor excludes such data or operations.
- e) Complex, matrix, statistical, or symbolic operations. This part neither requires nor excludes such data or operations.
- f) The properties of arithmetic datatypes that are not related to the numerical process, such as the representation of values on physical media.
- g) 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,
 - 2) the syntax used for parsed (input) or generated (output) character string forms for numerals by any specific programming language or library,
 - 3) the precedence of operators in the programming language,
 - 4) the presence or absence of automatic datatype coercions,
 - 5) the rules for assignment, parameter passing, and returning value,
 - 6) the consequences of applying an operation to values of improper datatype, or to uninitialised data.

Furthermore, this part 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 parameters and operations specified in this part and the concrete language syntax of some programming language. More generally, a binding standard specifies the correspondence between certain 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 if and only if it conforms to the binding standard. In case of conflict between a binding standard and this part, the specification of the binding standard takes precedence.

When a binding standard covers only a subset of the operations specified in this part, an implementation remains free to conform to this part with respect to other operations, independently of that binding standard.

When no binding standard for a language and some operations specified in this part exists, an implementation conforms to this part if and only if it provides one or more operations that together satisfy all the requirements of clauses 5 through 8 that are relevant to those operations. The implementation shall then document the binding.

Conformity to this part is always with respect to a specified set of datatypes and operations. Conformity to this part implies conformity to part 1 for the integer and floating point datatypes used.

An implementation is free to provide operations that do not conform to this part, or that are beyond the scope of this part. The implementation shall not claim or imply conformity to this part with respect to such operations.

An implementation is permitted to have modes of operation that do not conform to this part. 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 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 for suggested language bindings.
- 2 A complete binding for this part will include (explicitly or by reference) a binding for part 1 as well, which in turn may include (explicitly or by reference) a binding for IEC 60559 as well.
- 3 This part does not require a particular set of operations to be provided. It is not possible to conform to this part without specifying to which datatypes and set of operations (and modes of operation) conformity is claimed.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 10967. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 10967 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems*.

ISO/IEC 10967-1:1994, *Information technology – Language independent arithmetic – Part 1: Integer and floating point arithmetic*.

NOTE – See also annex E.

4 Symbols and definitions

4.1 Symbols

4.1.1 Sets and intervals

In this part, $\mathcal{Z}$ denotes the set of mathematical integers, $\mathcal{R}$ denotes the set of classical 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 manipulation is used.

In this part, the following notation for intervals is used

$[x, z]$ 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.

4.1.2 Operators and relations

All prefix and infix operators have their conventional (exact) mathematical meaning. In particular this part uses

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

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 :

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

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

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

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

NOTE – $i|j$ is true exactly when j/i is defined and $j/i \in \mathbb{Z}$.

4.1.3 Mathematical functions

This part specifies properties for a number of operations numerically approximating some of the elementary functions. The following ideal mathematical functions are defined in chapter 4 of the *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* [47] (e is the Napierian base)

$$e^x, x^y, \sqrt{x}, \ln, \log_b,$$

$$\sin, \cos, \tan, \cot, \sec, \csc, \arcsin, \arccos, \arctan, \text{arccot}, \text{arcsec}, \text{arccsc},$$

$$\sinh, \cosh, \tanh, \coth, \text{sech}, \text{csch}, \text{arcsinh}, \text{arccosh}, \text{arctanh}, \text{arcoth}, \text{arcsech}, \text{arccsch}.$$

Many of the inverses above are multi-valued. The selection of which value to return, the principal value, so as to make the inverses into functions, is done in the conventional way. E.g., $\sqrt{x} \in [0, \infty[$ when $x \in [0, \infty[$. The only one over which there is some difference of conventions is the arccot function. Conventions there vary for negative arguments; either a negative value (giving a sign symmetric function), or a positive return value (giving a function that is continuous over zero). In this part, arccot refers to the sign symmetric inverse function (with a branch cut at 0), and arccotc refers to the continuous inverse function.

$$\text{arccosh}(x) \geq 0, \text{arcsech}(x) \geq 0,$$

$$\arcsin(x) \in [-\pi/2, \pi/2], \arccos(x) \in [0, \pi], \arctan(x) \in]-\pi/2, \pi/2[,$$

$$\text{arccot}(x) \in]-\pi/2, \pi/2], \text{arccotc}(x) \in]0, \pi[, \text{arcsec}(x) \in [0, \pi], \text{arccsc}(x) \in [-\pi/2, \pi/2].$$

NOTE – $e = 2.71828\dots$ e is not in any floating point datatype conforming to part 1, unless added as a special value, which is usually not done.

4.1.4 Exceptional values

The exceptional value **underflow** is used in this part as it is in part 1.

Three new exceptional values, **overflow**, **invalid**, and **infinitary**, are introduced in this part replacing three other exceptional values used in part 1. **invalid** and **infinitary** are in this part used instead of the **undefined** of part 1. **overflow** is used instead of the **integer_overflow** and **floating_overflow** of part 1. Bindings may still distinguish between **integer_overflow** and **floating_overflow**.

One new exceptional value, **absolute_precision_underflow**, is introduced in this part with no correspondence in part 1. The exceptional value **absolute_precision_underflow** is used when the given floating point angle value argument is so big that even a highly accurate result from a trigonometric operation is questionable, due to the fact that the density of floating point values has decreased significantly at these big angle values.

For the exceptional values, a continuation value may be given in parenthesis after the exceptional value.

4.1.5 Datatypes

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

NOTE 1 – Mathematical relations are true or false (or undefined, if an operand is undefined). In contrast, **true** and **false** are values in **Boolean**.

For pairs, define:

$$\begin{aligned}fst((x, y)) &= x \\snd((x, y)) &= y\end{aligned}$$

Square brackets are used to write finite sequences of values. $[]$ is the sequence containing no values. $[s]$, is the sequence of one value, s . $[s_1, s_2]$, is the sequence of two values, s_1 and then s_2 , etc. The colon operator is used to prepend a value to a sequence: $x : [x_1, \dots, x_n] = [x, x_1, \dots, x_n]$. $[S]$, where S is a set, denotes the set of finite sequences, where each value in a sequence is in S .

NOTE 2 – It is always clear from context, in the text of this part, if $[X]$ is a sequence of one element, or the set of sequences with values from X . It is also clear from context if $[x_1, x_2]$ is a sequence of two values or an interval.

Integer datatypes and floating point datatypes are defined in part 1. Let I be the non-special value set for an integer datatype conforming to part 1. Let F be the non-special value set for a floating point datatype conforming to part 1. Floating point datatypes that conform to part 1 shall, for use with this part, have a value for the parameter p_F such that $p_F \geq 2 \cdot \max\{1, \lceil \log_{r_F}(2 \cdot \pi) \rceil\}$, and have a value for the parameter $emin_F$ such that $emin_F \leq -p_F - 1$.

NOTES

- 3 This implies that $fminN_F < 0.5 \cdot \epsilon_F / r_F$ in this part, rather than just $fminN_F \leq \epsilon_F$.
- 4 These extra requirements, which do not limit the use of any existing floating point datatype, are made 1) so that angles in radians are not too degenerate within the first two cycles, plus and minus, when represented in F , and 2) in order to be able to avoid the underflow notification in specifications for the $expm1_F$ and $ln1p_F$ operations.
- 5 F should also be such that $p_F \geq 2 + \lceil \log_{r_F}(1000) \rceil$, to allow for a not too coarse angle resolution anywhere in the interval $[-big_angle_{r_F}, big_angle_{r_F}]$. See clause 5.3.8.

The following symbols represent special values defined in IEC 60559 and used in this part:

-0, **+\infty**, **-\infty**, **qNaN**, and **sNaN**.

These values are not part of the set I or the set F , but if iec_559_F has the value **true**, these values are included in the floating point datatype corresponding to F .

NOTE 6 – This part 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 cover the results to be returned by an operation if given one or more of the IEC 60559 special values **-0**, **+\infty**, **-\infty**, 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, the actual result or continuation value shall be 0. If an implementation is not capable of representing a prescribed result or continuation value

of the IEC 60559 special values $+\infty$, $-\infty$, or **qNaN**, the actual result or continuation value is binding or implementation defined.

The following symbols used in this part are defined in part 1:

Integer parameters:

bounded_I, *maxint_I*, and *minint_I*.

Integer helper function:

wrap_I.

Integer operations:

neg_I, *add_I*, *sub_I*, and *mul_I*.

Floating point parameters:

r_F, *p_F*, *emin_F*, *emax_F*, *denorm_F*, and *iec_559_F*.

Derived floating point constants:

fmax_F, *fmin_F*, *fminN_F*, *fminD_F*, and *epsilon_F*.

Floating point rounding constant:

rnd_error_F.

Floating point value sets related to *F*:

*F**, *F_D*, and *F_N*.

Floating point helper functions:

e_F, *result_F*, and *rnd_F*.

Floating point operations:

neg_F, *add_F*, *sub_F*, *mul_F*, *div_F*, and *ulp_F*.

4.2 Definitions of terms

For the purposes of this part, the following definitions apply:

accuracy: The closeness between the true mathematical result and a computed result.

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

NOTE 1 – This part specifies requirements for integer and floating point datatypes. Complex numbers are not covered by this part, but will be included in a subsequent part of ISO/IEC 10967 [3].

continuation value: A 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 an IEC 60559 special value. (Contrast with *exceptional value*. See clause 6.1 of part 1.)

denormalisation loss: A larger than normal rounding error caused by the fact that subnormal values have less than full precision. (See clause 5.2 of part 1 for a full definition.)

error: (1) The difference between a computed value and the correct value. (Used in phrases like “rounding error” or “error bound”.)

(2) A synonym for *exception* in phrases like “error message” or “error output”. Error and exception are not synonyms in any other context.

exception: The 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**).

NOTE 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 C.

exceptional value: A non-numeric value produced by an arithmetic operation to indicate the occurrence of an exception. Exceptional values are not used in subsequent arithmetic processing. (See clause 5 of part 1.)

NOTES

- 3 Exceptional values are used as part of the defining formalism only. With respect to this part, 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.

helper function: A function used solely to aid in the expression of a requirement. Helper functions are not visible to the programmer, and are not required to be part of an implementation. However, some implementation defined helper functions are required to be documented.

implementation (of this part): The total arithmetic environment presented to a programmer, including hardware, language processors, exception handling facilities, subroutine libraries, other software, and documentation pertinent to this part.

literal: A syntactic entity, that does not have any proper sub-entity that is an expression, denoting a constant value.

monotonic approximation: An approximation helper function $h : \dots \times S \times \dots \rightarrow \mathcal{R}$, where the other arguments are kept constant, and where $S \subseteq \mathcal{R}$, is a monotonic approximation of a predetermined mathematical function $f : \mathcal{R} \rightarrow \mathcal{R}$ if, for every $a \in S$ and $b \in S$, where $a < b$,

- a) f is monotonic non-decreasing on $[a, b]$ implies $h(\dots, a, \dots) \leq h(\dots, b, \dots)$,
- b) f is monotonic non-increasing on $[a, b]$ implies $h(\dots, a, \dots) \geq h(\dots, b, \dots)$.

monotonic non-decreasing: A function $f : \mathcal{R} \rightarrow \mathcal{R}$ is monotonic non-decreasing on a real interval $[a, b]$ if for every x and y such that $a \leq x \leq y \leq b$, $f(x)$ and $f(y)$ are well-defined and $f(x) \leq f(y)$.

monotonic non-increasing: A function $f : \mathcal{R} \rightarrow \mathcal{R}$ is monotonic non-increasing on a real interval $[a, b]$ if for every x and y such that $a \leq x \leq y \leq b$, $f(x)$ and $f(y)$ are well-defined and $f(x) \geq f(y)$.

normalised: The non-zero values of a floating point type F that provide the full precision allowed by that type. (See F_N in clause 5.2 of part 1 for a full definition.)

notification: The process by which a program (or that program’s end user) is informed that an arithmetic exception has occurred. For example, dividing 2 by 0 results in a notification. (See clause 6 of part 1 for details.)

numeral: A numeric literal. It may denote a value in $\mathcal{Z}$ or $\mathcal{R}$, $-\mathbf{0}$, an infinity, or a NaN.

numerical function: A computer routine or other mechanism for the approximate evaluation of a mathematical function.

operation: A function directly available to the programmer, as opposed to helper functions or theoretical mathematical functions.

pole: A mathematical function f has a pole at x_0 if x_0 is finite, f is defined, finite, monotone, and continuous in at least one side of the neighbourhood of x_0 , and $\lim_{x \rightarrow x_0} f(x)$ is infinite.

precision: The number of digits in the fraction of a floating point number. (See clause 5.2 of part 1.)

rounding: The act of computing a representable final result for an operation that is close to the exact (but unrepresentable) result for that operation. Note that a suitable representable result may not exist (see clause 5.2 of part 1).

rounding function: Any 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 $\mathcal{R}$,

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

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

round to nearest: The property of a rounding function rnd that when $u \in \mathcal{R}$ is between two adjacent values in X , $rnd(u)$ selects the one nearest u . If the adjacent values are equidistant from u , either may be chosen deterministically, but so that $rnd(-u) = -rnd(u)$.

round toward minus infinity: The property of a rounding function rnd that when $u \in \mathcal{R}$ is between two adjacent values in X , $rnd(u)$ selects the one less than u .

round toward plus infinity: The property of a rounding function rnd that when $u \in \mathcal{R}$ is between two adjacent values in X , $rnd(u)$ selects the one greater than u .

shall: A verbal form used to indicate requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted. (Quoted from the directives [1].)

should: A verbal form used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that (in the negative form) a certain possibility is deprecated but not prohibited. (Quoted from the directives [1].)

signature (of a function or operation): A 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. Approximation helper functions may be undefined for some argument values.

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.

subnormal: The non-zero values of a floating point type F that provide less than the full precision allowed by that type. (See F_D in clause 5.2 of part 1 for a full definition. In part 1 and IEC 60559 this concept is called denormal.)

ulp: The value of one “unit in the last place” of a floating point number. This value depends on the exponent, the radix, and the precision used in representing the number. Thus, the ulp of a normalised value x (in F), with exponent t , precision p_F , and radix r_F , is $r_F^{t-p_F}$, and the ulp of a subnormal or zero value is $fminD_F$. (See clause 5.2 of part 1.)

5 Specifications for integer and floating point operations

This clause specifies a number of helper functions and operations for integer and floating point datatypes. Each operation is given a signature and is further specified by a number of cases. These cases may refer to other operations (specified in this part or in part 1), to mathematical functions, and to helper functions (specified in this part or in part 1). They also use special abstract values ($-\infty$, $+\infty$, -0 , **qNaN**, **sNaN**). For each datatype, two of these abstract values may represent several actual values each: **qNaN** and **sNaN**. Finally, the specifications may refer to exceptional values.

The signatures in the specifications in this clause specify only all non-special values as input values, and indicate as output values a superset of all non-special, special, and exceptional values that may result from these (non-special) input values. Exceptional and special values that can never result from non-special input values are not included in the signatures given. Also, signatures that, for example, include IEC 60559 special values as arguments are not given in the specifications below. This does not exclude such signatures from being valid for these operations.

5.1 Basic integer operations

Clause 5.1 of part 1 specifies integer datatypes and a number of operations on values of an integer datatype. In this clause some additional operations on values of an integer datatype are specified.

I is the set of non-special values, $I \subseteq \mathcal{Z}$, for an integer datatype conforming to part 1. Integer datatypes conforming to part 1 often do not contain any NaN or infinity values, even though they may do so. Therefore this clause has no specifications for such values as arguments or results other than as continuation values.

NOTE – For some integer operations, **infinitary** notifications may occur. For **infinitary** notifications, an infinitary continuation value is recommended. For bounded integer datatypes, $maxint_I$ or $minint_I$ may be used as replacement continuation values as appropriate, if infinitary values are not available in the datatype. For unbounded integer datatypes, however, no $maxint_I$ and $minint_I$ in I are defined, and infinitary values should be used.

5.1.1 The integer *result* and *wrap* helper functions

The $result_I$ helper function:

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

$$\begin{aligned} result_I(x) &= x && \text{if } x \in I \\ &= \mathbf{overflow} && \text{if } x \in \mathcal{Z} \text{ and } x \notin I \end{aligned}$$

The $wrap_I$ helper function:

$$\begin{aligned} wrap_I : \mathcal{Z} &\rightarrow I \\ wrap_I(x) &= x && \text{if } x \in I \\ &= x - (n \cdot (maxint_I - minint_I + 1)) && \text{if } x \in \mathcal{Z} \text{ and } x \notin I \end{aligned}$$

where $n \in \mathcal{Z}$ is chosen such that the result is in I .

NOTES

- 1 $n = \lfloor (x - minint_I) / (maxint_I - minint_I + 1) \rfloor$ if $x \in \mathcal{Z}$ and $bounded_I = \mathbf{true}$; or equivalently $n = \lceil (x - maxint_I) / (maxint_I - minint_I + 1) \rceil$ if $x \in \mathcal{Z}$ and $bounded_I = \mathbf{true}$.
- 2 For some wrapping basic arithmetic operations this n is computed by the ‘*ov*’ operations in clause 5.1.9.
- 3 The $wrap_I$ helper function is also used in part 1.

5.1.2 Integer maximum and minimum

$$\begin{aligned} max_I : I \times I &\rightarrow I \\ max_I(x, y) &= \max\{x, y\} && \text{if } x, y \in I \end{aligned}$$

$$\begin{aligned} min_I : I \times I &\rightarrow I \\ min_I(x, y) &= \min\{x, y\} && \text{if } x, y \in I \end{aligned}$$

$$\begin{aligned} max_seq_I : [I] &\rightarrow I \cup \{\mathbf{infinitary}\} \\ max_seq_I([x_1, \dots, x_n]) &= \mathbf{infinitary}(-\infty) && \text{if } n = 0 \\ &= \max\{x_1, \dots, x_n\} && \text{if } n \geq 1 \text{ and } \{x_1, \dots, x_n\} \subseteq I \end{aligned}$$

$$\begin{aligned} min_seq_I : [I] &\rightarrow I \cup \{\mathbf{infinitary}\} \\ min_seq_I([x_1, \dots, x_n]) &= \mathbf{infinitary}(+\infty) && \text{if } n = 0 \\ &= \min\{x_1, \dots, x_n\} && \text{if } n \geq 1 \text{ and } \{x_1, \dots, x_n\} \subseteq I \end{aligned}$$

5.1.3 Integer diminish

$$\begin{aligned} dim_I : I \times I &\rightarrow I \cup \{\mathbf{overflow}\} \\ dim_I(x, y) &= result_I(\max\{0, x - y\}) && \text{if } x, y \in I \end{aligned}$$

NOTE – dim_I cannot be implemented as $max_I(0, sub_I(x, y))$ for bounded integer types, since this latter expression has other overflow properties.

5.1.4 Integer power and arithmetic shift

$$\begin{aligned}
 & power_I : I \times I \rightarrow I \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\} \\
 & power_I(x, y) = result_I(x^y) && \text{if } x, y \in I \text{ and } (y > 0 \text{ or } |x| = 1) \\
 & \quad = 1 && \text{if } x \in I \text{ and } x \neq 0 \text{ and } y = 0 \\
 & \quad = \mathbf{invalid}(1) && \text{if } x = 0 \text{ and } y = 0 \\
 & \quad = \mathbf{infinitary}(+\infty) && \text{if } x = 0 \text{ and } y \in I \text{ and } y < 0 \\
 & \quad = \mathbf{invalid}(0) && \text{if } x, y \in I \text{ and } x \notin \{-1, 0, 1\} \text{ and } y < 0
 \end{aligned}$$

$$\begin{aligned}
 & shift2_I : I \times I \rightarrow I \cup \{\mathbf{overflow}\} \\
 & shift2_I(x, y) = result_I(\lfloor x \cdot 2^y \rfloor) && \text{if } x, y \in I
 \end{aligned}$$

$$\begin{aligned}
 & shift10_I : I \times I \rightarrow I \cup \{\mathbf{overflow}\} \\
 & shift10_I(x, y) = result_I(\lfloor x \cdot 10^y \rfloor) && \text{if } x, y \in I
 \end{aligned}$$

5.1.5 Integer square root

$$\begin{aligned}
 & sqrt_I : I \rightarrow I \cup \{\mathbf{invalid}\} \\
 & sqrt_I(x) = \lfloor \sqrt{x} \rfloor && \text{if } x \in I \text{ and } x \geq 0 \\
 & \quad = \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \in I \text{ and } x < 0
 \end{aligned}$$

5.1.6 Divisibility tests

$$\begin{aligned}
 & divides_I : I \times I \rightarrow \mathbf{Boolean} \\
 & divides_I(x, y) = \mathbf{true} && \text{if } x, y \in I \text{ and } x|y \\
 & \quad = \mathbf{false} && \text{if } x, y \in I \text{ and not } x|y
 \end{aligned}$$

NOTES

- 1 $divides_I(0, 0) = \mathbf{false}$, since 0 does not divide anything, not even 0.
- 2 $divides_I$ cannot be implemented as, e.g., $eq_I(0, mod_I(y, x))$, since the remainder functions give notifications for a zero second argument.

$$\begin{aligned}
 & even_I : I \rightarrow \mathbf{Boolean} \\
 & even_I(x) = \mathbf{true} && \text{if } x \in I \text{ and } 2|x \\
 & \quad = \mathbf{false} && \text{if } x \in I \text{ and not } 2|x
 \end{aligned}$$

$$\begin{aligned}
 & odd_I : I \rightarrow \mathbf{Boolean} \\
 & odd_I(x) = \mathbf{true} && \text{if } x \in I \text{ and not } 2|x \\
 & \quad = \mathbf{false} && \text{if } x \in I \text{ and } 2|x
 \end{aligned}$$

5.1.7 Integer division (with floor, round, or ceiling) and remainder

$quot_I : I \times I \rightarrow I \cup \{\text{overflow}, \text{infinitary}, \text{invalid}\}$

$$\begin{aligned} quot_I(x, y) &= result_I(\lfloor x/y \rfloor) && \text{if } x, y \in I \text{ and } y \neq 0 \\ &= \text{infinitary}(+\infty) && \text{if } x \in I \text{ and } x > 0 \text{ and } y = 0 \\ &= \text{invalid}(\text{qNaN}) && \text{if } x = 0 \text{ and } y = 0 \\ &= \text{infinitary}(-\infty) && \text{if } x \in I \text{ and } x < 0 \text{ and } y = 0 \end{aligned}$$

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

$mod_I : I \times I \rightarrow I \cup \{\text{invalid}\}$

$$\begin{aligned} mod_I(x, y) &= x - (\lfloor x/y \rfloor \cdot y) && \text{if } x, y \in I \text{ and } y \neq 0 \\ &= \text{invalid}(\text{qNaN}) && \text{if } x \in I \text{ and } y = 0 \end{aligned}$$

$ratio_I : I \times I \rightarrow I \cup \{\text{overflow}, \text{infinitary}, \text{invalid}\}$

$$\begin{aligned} ratio_I(x, y) &= result_I(\text{round}(x/y)) && \text{if } x, y \in I \text{ and } y \neq 0 \\ &= \text{infinitary}(+\infty) && \text{if } x \in I \text{ and } x > 0 \text{ and } y = 0 \\ &= \text{invalid}(\text{qNaN}) && \text{if } x = 0 \text{ and } y = 0 \\ &= \text{infinitary}(-\infty) && \text{if } x \in I \text{ and } x < 0 \text{ and } y = 0 \end{aligned}$$

$residue_I : I \times I \rightarrow I \cup \{\text{overflow}, \text{invalid}\}$

$$\begin{aligned} residue_I(x, y) &= result_I(x - (\text{round}(x/y) \cdot y)) \\ &= \text{invalid}(\text{qNaN}) && \text{if } x \in I \text{ and } y = 0 \end{aligned}$$

$group_I : I \times I \rightarrow I \cup \{\text{overflow}, \text{infinitary}, \text{invalid}\}$

$$\begin{aligned} group_I(x, y) &= result_I(\lceil x/y \rceil) && \text{if } x, y \in I \text{ and } y \neq 0 \\ &= \text{infinitary}(+\infty) && \text{if } x \in I \text{ and } x > 0 \text{ and } y = 0 \\ &= \text{invalid}(\text{qNaN}) && \text{if } x = 0 \text{ and } y = 0 \\ &= \text{infinitary}(-\infty) && \text{if } x \in I \text{ and } x < 0 \text{ and } y = 0 \end{aligned}$$

$pad_I : I \times I \rightarrow I \cup \{\text{invalid}\}$

$$\begin{aligned} pad_I(x, y) &= (\lceil x/y \rceil \cdot y) - x && \text{if } x, y \in I \text{ and } y \neq 0 \\ &= \text{invalid}(\text{qNaN}) && \text{if } x \in I \text{ and } y = 0 \end{aligned}$$

5.1.8 Greatest common divisor and least common positive multiple

$gcd_I : I \times I \rightarrow I \cup \{\text{overflow}, \text{infinitary}\}$

$$\begin{aligned} gcd_I(x, y) &= result_I(\max\{v \in \mathcal{Z} \mid v|x \text{ and } v|y\}) \\ &= \text{infinitary}(+\infty) && \text{if } x, y \in I \text{ and } (x \neq 0 \text{ or } y \neq 0) \\ &= \text{infinitary}(+\infty) && \text{if } x = 0 \text{ and } y = 0 \end{aligned}$$

$lcm_I : I \times I \rightarrow I \cup \{\text{overflow}\}$

$$\begin{aligned}
lcm_I(x, y) &= result_I(\min\{v \in \mathcal{Z} \mid x|v \text{ and } y|v \text{ and } v > 0\}) \\
&\quad \text{if } x, y \in I \text{ and } x \neq 0 \text{ and } y \neq 0 \\
&= 0 \quad \text{if } x, y \in I \text{ and } (x = 0 \text{ or } y = 0)
\end{aligned}$$

$$\begin{aligned}
gcd_seq_I : [I] &\rightarrow I \cup \{\mathbf{overflow}, \mathbf{infinitary}\} \\
gcd_seq_I([x_1, \dots, x_n]) &= result_I(\max\{v \in \mathcal{Z} \mid v|x_i \text{ for all } i \in \{1, \dots, n\}\}) \\
&\quad \text{if } \{x_1, \dots, x_n\} \subseteq I \text{ and } \{x_1, \dots, x_n\} \not\subseteq \{0\} \\
&= \mathbf{infinitary}(+\infty) \quad \text{if } \{x_1, \dots, x_n\} \subseteq \{0\}
\end{aligned}$$

$$\begin{aligned}
lcm_seq_I : [I] &\rightarrow I \cup \{\mathbf{overflow}\} \\
lcm_seq_I([x_1, \dots, x_n]) &= result_I(\min\{v \in \mathcal{Z} \mid x_i|v \text{ for all } i \in \{1, \dots, n\} \text{ and } v > 0\}) \\
&\quad \text{if } \{x_1, \dots, x_n\} \subseteq I \text{ and } 0 \notin \{x_1, \dots, x_n\} \\
&= 0 \quad \text{if } \{x_1, \dots, x_n\} \subseteq I \text{ and } 0 \in \{x_1, \dots, x_n\}
\end{aligned}$$

NOTE – These specifications imply: $gcd_seq_I([]) = \mathbf{infinitary}(+\infty)$ and $lcm_seq_I([]) = 1$.

5.1.9 Support operations for extended integer range

These operations can be used to implement extended range integer datatypes, including unbounded integer datatypes.

$$\begin{aligned}
add_wrap_I : I \times I &\rightarrow I \\
add_wrap_I(x, y) &= wrap_I(x + y) \quad \text{if } x, y \in I
\end{aligned}$$

$$\begin{aligned}
add_ov_I : I \times I &\rightarrow \{-1, 0, 1\} \\
add_ov_I(x, y) &= ((x + y) - add_wrap_I(x, y)) / (maxint_I - minint_I + 1) \\
&\quad \text{if } x, y \in I \text{ and } bounded_I = \mathbf{true} \\
&= 0 \quad \text{if } x, y \in I \text{ and } bounded_I = \mathbf{false}
\end{aligned}$$

$$\begin{aligned}
sub_wrap_I : I \times I &\rightarrow I \\
sub_wrap_I(x, y) &= wrap_I(x - y) \quad \text{if } x, y \in I
\end{aligned}$$

$$\begin{aligned}
sub_ov_I : I \times I &\rightarrow \{-1, 0, 1\} \\
sub_ov_I(x, y) &= ((x - y) - sub_wrap_I(x, y)) / (maxint_I - minint_I + 1) \\
&\quad \text{if } x, y \in I \text{ and } bounded_I = \mathbf{true} \\
&= 0 \quad \text{if } x, y \in I \text{ and } bounded_I = \mathbf{false}
\end{aligned}$$

$$\begin{aligned}
mul_wrap_I : I \times I &\rightarrow I \\
mul_wrap_I(x, y) &= wrap_I(x \cdot y) \quad \text{if } x, y \in I
\end{aligned}$$

$$mul_{ov_I} : I \times I \rightarrow I$$

$$mul_{ov_I}(x, y) = ((x \cdot y) - mul_{wrap_I}(x, y)) / (maxint_I - minint_I + 1)$$

$$= 0 \quad \begin{array}{l} \text{if } x, y \in I \text{ and } bounded_I = \mathbf{true} \\ \text{if } x, y \in I \text{ and } bounded_I = \mathbf{false} \end{array}$$

NOTE – The add_{ov_I} and sub_{ov_I} will only return -1 (for negative overflow), 0 (no overflow), and 1 (for positive overflow).

5.2 Basic floating point operations

Clause 5.2 of part 1 specifies floating point datatypes and a number of operations on values of a floating point datatype. In this clause some additional operations on values of a floating point datatype are specified.

NOTE – Further operations on values of a floating point datatype, for elementary floating point numerical functions, are specified in clause 5.3.

F is the non-special value set, $F \subset \mathcal{R}$, for a floating point datatype conforming to part 1. Floating point datatypes conforming to part 1 often do contain $-\mathbf{0}$, infinity, and NaN values. Therefore, in this clause there are specifications for such values as arguments.

5.2.1 The rounding and floating point *result* helper functions

Floating point rounding helper functions (F^* is defined in part 1):

The floating point helper function

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

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

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

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

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

is the rounding function that rounds to nearest. $nearest_F$ is partially implementation defined: the handling of ties is implementation defined, but must be sign symmetric. If $iec_559_F = \mathbf{true}$, the semantics of $nearest_F$ is completely defined by IEC 60559: in this case ties are rounded so that the result has an even last digit.

$result_F$ is a helper function that is partially implementation defined. $result_F$ has a signature:

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

For the overflow cases it is defined as:

$$\begin{array}{ll} result_F(x, nearest_F) = \mathbf{overflow}(+\infty) & \text{if } x \in \mathcal{R} \text{ and } nearest_F(x) > fmax_F \\ result_F(x, nearest_F) = \mathbf{overflow}(-\infty) & \text{if } x \in \mathcal{R} \text{ and } nearest_F(x) < -fmax_F \\ result_F(x, up_F) = \mathbf{overflow}(+\infty) & \text{if } x \in \mathcal{R} \text{ and } up_F(x) > fmax_F \\ result_F(x, up_F) = \mathbf{overflow}(-fmax_F) & \text{if } x \in \mathcal{R} \text{ and } up_F(x) < -fmax_F \\ result_F(x, down_F) = \mathbf{overflow}(fmax_F) & \text{if } x \in \mathcal{R} \text{ and } down_F(x) > fmax_F \\ result_F(x, down_F) = \mathbf{overflow}(-\infty) & \text{if } x \in \mathcal{R} \text{ and } down_F(x) < -fmax_F \end{array}$$

For other cases and for any rounding function rnd in $(\mathcal{R} \rightarrow F^*)$, the following shall apply:

$$\begin{aligned}
 result_F(x, rnd) &= x && \text{if } x = 0 \\
 &= rnd(x) && \text{if } x \in \mathcal{R} \text{ and } fminN_F \leq |x| \text{ and } |rnd(x)| \leq fmax_F \\
 &= rnd(x) \text{ or } \mathbf{underflow}(c) && \text{if } x \in \mathcal{R} \text{ and } |x| < fminN_F \text{ and } |rnd(x)| = fminN_F \\
 &&& \text{and } rnd \text{ has no denormalisation loss at } x \\
 &= rnd(x) \text{ or } \mathbf{underflow}(c) && \text{if } x \in \mathcal{R} \text{ and } denorm_F = \mathbf{true} \text{ and} \\
 &&& |rnd(x)| < fminN_F \text{ and } x \neq 0 \\
 &&& \text{and } rnd \text{ has no denormalisation loss at } x \\
 &= \mathbf{underflow}(c) && \text{otherwise}
 \end{aligned}$$

where

$$\begin{aligned}
 c &= rnd(x) && \text{when } denorm_F = \mathbf{true} \text{ and } (rnd(x) \neq 0 \text{ or } x > 0), \\
 c &= -\mathbf{0} && \text{when } denorm_F = \mathbf{true} \text{ and } rnd(x) = 0 \text{ and } x < 0, \\
 c &= 0 && \text{when } denorm_F = \mathbf{false} \text{ and } x > 0, \\
 c &= -\mathbf{0} && \text{when } denorm_F = \mathbf{false} \text{ and } x < 0
 \end{aligned}$$

An implementation is allowed to choose between $rnd(x)$ and $\mathbf{underflow}(rnd(x))$ in the region between 0 and $fminN_F$. However, a subnormal value without an underflow notification can be chosen only if $denorm_F = \mathbf{true}$ and no denormalisation loss occurs at x .

NOTES

- 1 This differs from the specification of $result_F$ as given in part 1 in the following respects:
1) the continuation values on overflow and underflow are given directly here, and 2) all instances of denormalisation loss must be accompanied with an underflow notification.
- 2 $denorm_F = \mathbf{false}$ implies $iec_559_F = \mathbf{false}$, and $iec_559_F = \mathbf{true}$ implies $denorm_F = \mathbf{true}$.
- 3 If $iec_559_F = \mathbf{true}$, then subnormal or zero results that have no denormalisation loss do not result in an underflow notification, if the notification is by recording of indicators.

Define the no_result_F , $no_result2_F$, and $no_result3_F$ helper functions:

$$no_result_F : F \rightarrow \{\mathbf{invalid}\}$$

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

$$no_result2_F : F \times F \rightarrow \{\mathbf{invalid}\}$$

$$\begin{aligned}
 no_result2_F(x, y) &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x, y \in F \cup \{-\infty, -\mathbf{0}, +\infty\} \\
 &= \mathbf{qNaN} && \text{if at least one of } x \text{ and } y \text{ is a quiet NaN and} \\
 &&& \text{neither a signalling NaN} \\
 &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN or } y \text{ is a signalling NaN}
 \end{aligned}$$

$$no_result3_F : F \times F \times F \rightarrow \{\mathbf{invalid}\}$$

$$\begin{aligned}
 no_result3_F(x, y, z) &= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x, y, z \in F \cup \{-\infty, -\mathbf{0}, +\infty\} \\
 &= \mathbf{qNaN} && \text{if at least one of } x, y, \text{ and } z \text{ is a quiet NaN and} \\
 &&& \text{neither is a signalling NaN}
 \end{aligned}$$

$= \text{invalid}(\text{qNaN})$ if at least one of x , y , or z is a signalling NaN

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 4 – The handling of other special values, if available, is left unspecified by this part.

5.2.2 Floating point maximum and minimum

The appropriate return value of the maximum and minimum operations given a quiet NaN (**qNaN**) as one of the input values depends on the circumstances for each point of use. Sometimes **qNaN** is the appropriate result, sometimes the non-NaN argument is the appropriate result. Therefore, two variants each of the floating point maximum and minimum operations are specified here, and the programmer can decide which one is appropriate to use at each particular place of usage, assuming both variants are included in the binding.

$\max_F : F \times F \rightarrow F$

$\max_F(x, y)$	$= \max\{x, y\}$	if $x, y \in F$
	$= +\infty$	if $x = +\infty$ and $y \in F \cup \{-\infty, -0\}$
	$= y$	if $x = -0$ and $y \in F$ and $y \geq 0$
	$= -0$	if $x = -0$ and $((y \in F \text{ and } y < 0) \text{ or } y = -0)$
	$= y$	if $x = -\infty$ and $y \in F \cup \{+\infty, -0\}$
	$= +\infty$	if $y = +\infty$ and $x \in F \cup \{+\infty, -0\}$
	$= x$	if $y = -0$ and $x \in F$ and $x \geq 0$
	$= -0$	if $y = -0$ and $x \in F$ and $x < 0$
	$= x$	if $y = -\infty$ and $x \in F \cup \{-\infty, -0\}$
	$= \text{no_result2}_F(x, y)$	otherwise

$\min_F : F \times F \rightarrow F$

$\min_F(x, y)$	$= \min\{x, y\}$	if $x, y \in F$
	$= y$	if $x = +\infty$ and $y \in F \cup \{-\infty, -0\}$
	$= -0$	if $x = -0$ and $y \in F$ and $y \geq 0$
	$= y$	if $x = -0$ and $((y \in F \text{ and } y < 0) \text{ or } y = -0)$
	$= -\infty$	if $x = -\infty$ and $y \in F \cup \{+\infty, -0\}$
	$= x$	if $y = +\infty$ and $x \in F \cup \{+\infty, -0\}$
	$= -0$	if $y = -0$ and $x \in F$ and $x \geq 0$
	$= x$	if $y = -0$ and $x \in F$ and $x < 0$
	$= -\infty$	if $y = -\infty$ and $x \in F \cup \{-\infty, -0\}$
	$= \text{no_result2}_F(x, y)$	otherwise

$m\max_F : F \times F \rightarrow F$

$m\max_F(x, y)$	$= \max_F(x, y)$	if $x, y \in F \cup \{+\infty, -0, -\infty\}$
	$= x$	if $x \in F \cup \{+\infty, -0, -\infty\}$ and y is a quiet NaN
	$= y$	if $y \in F \cup \{+\infty, -0, -\infty\}$ and x is a quiet NaN
	$= \text{no_result2}_F(x, y)$	otherwise

$m\min_F : F \times F \rightarrow F$

$$\begin{aligned}
mmin_F(x, y) &= min_F(x, y) && \text{if } x, y \in F \cup \{+\infty, -0, -\infty\} \\
&= x && \text{if } x \in F \cup \{+\infty, -0, -\infty\} \text{ and } y \text{ is a quiet NaN} \\
&= y && \text{if } y \in F \cup \{+\infty, -0, -\infty\} \text{ and } x \text{ is a quiet NaN} \\
&= no_result_F(x, y) && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
max_seq_F : [F] &\rightarrow F \cup \{\mathbf{infinitary}\} \\
max_seq_F([x_1, \dots, x_n]) &= \mathbf{infinitary}(-\infty) && \text{if } n = 0 \\
&= x_1 && \text{if } n = 1 \text{ and } x_1 \text{ is not a NaN} \\
&= max_F(max_seq_F([x_1, \dots, x_{n-1}]), x_n) && \text{if } n \geq 2 \\
&= no_result_F(x_1) && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
min_seq_F : [F] &\rightarrow F \cup \{\mathbf{infinitary}\} \\
min_seq_F([x_1, \dots, x_n]) &= \mathbf{infinitary}(+\infty) && \text{if } n = 0 \\
&= x_1 && \text{if } n = 1 \text{ and } x_1 \text{ is not a NaN} \\
&= min_F(min_seq_F([x_1, \dots, x_{n-1}]), x_n) && \text{if } n \geq 2 \\
&= no_result_F(x_1) && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
mmax_seq_F : [F] &\rightarrow F \cup \{\mathbf{infinitary}\} \\
mmax_seq_F([x_1, \dots, x_n]) &= \mathbf{infinitary}(-\infty) && \text{if } n = 0 \\
&= x_1 && \text{if } n = 1 \text{ and } x_1 \text{ is not a NaN} \\
&= mmax_F(mmax_seq_F([x_1, \dots, x_{n-1}]), x_n) && \text{if } n \geq 2 \\
&= no_result_F(x_1) && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
mmin_seq_F : [F] &\rightarrow F \cup \{\mathbf{infinitary}\} \\
mmin_seq_F([x_1, \dots, x_n]) &= \mathbf{infinitary}(+\infty) && \text{if } n = 0 \\
&= x_1 && \text{if } n = 1 \text{ and } x_1 \text{ is not a NaN} \\
&= mmin_F(mmin_seq_F([x_1, \dots, x_{n-1}]), x_n) && \text{if } n \geq 2 \\
&= no_result_F(x_1) && \text{otherwise}
\end{aligned}$$

5.2.3 Floating point diminish

$$\begin{aligned}
dim_F : F \times F &\rightarrow F \cup \{\mathbf{underflow}, \mathbf{overflow}\} \\
dim_F(x, y) &= result_F(\max\{0, x - y\}, rnd_F) \\
&= -0 && \text{if } x, y \in F \\
&= dim_F(0, y) && \text{if } x = -0 \text{ and } y = 0 \\
&= dim_F(x, 0) && \text{if } x = -0 \text{ and } y \in F \cup \{-\infty, -0, +\infty\} \text{ and } y \neq 0 \\
&= dim_F(x, 0) && \text{if } y = -0 \text{ and } x \in F \cup \{-\infty, +\infty\}
\end{aligned}$$

$= +\infty$	if $x = +\infty$ and $y \in F \cup \{-\infty\}$
$= 0$	if $x = -\infty$ and $y \in F \cup \{+\infty\}$
$= 0$	if $y = +\infty$ and $x \in F$
$= +\infty$	if $y = -\infty$ and $x \in F$
$= no_result2_F(x, y)$	otherwise

NOTE – dim_F cannot be implemented by $max_F(-0, sub_F(x, y))$, since this latter expression has other overflow properties.

5.2.4 Floor, round, and ceiling

$floor_F : F \rightarrow F$	
$floor_F(x)$	$= \lfloor x \rfloor$ if $x \in F$
	$= x$ if $x \in \{-\infty, -0, +\infty\}$
	$= no_result_F(x)$ otherwise

$floor_rest_F : F \rightarrow F$	
$floor_rest_F(x)$	$= result_F(x - \lfloor x \rfloor, rnd_F)$ if $x \in F$
	$= 0$ if $x = -0$
	$= no_result_F(x)$ otherwise

$rounding_F : F \rightarrow F \cup \{-0\}$	
$rounding_F(x)$	$= round(x)$ if $x \in F$ and $(x \geq 0$ or $round(x) \neq 0)$
	$= -0$ if $x \in F$ and $x < 0$ and $round(x) = 0$
	$= x$ if $x \in \{-\infty, -0, +\infty\}$
	$= no_result_F(x)$ otherwise

NOTE – $round_F$ is a different operation specified in part 1.

$rounding_rest_F : F \rightarrow F$	
$rounding_rest_F(x)$	$= x - round(x)$ if $x \in F$
	$= 0$ if $x = -0$
	$= no_result_F(x)$ otherwise

$ceiling_F : F \rightarrow F \cup \{-0\}$	
$ceiling_F(x)$	$= \lceil x \rceil$ if $x \in F$ and $(x \geq 0$ or $\lceil x \rceil \neq 0)$
	$= -0$ if $x \in F$ and $x < 0$ and $\lceil x \rceil = 0$
	$= x$ if $x \in \{-\infty, -0, +\infty\}$
	$= no_result_F(x)$ otherwise

$ceiling_rest_F : F \rightarrow F$

$$\begin{aligned}
\text{ceiling_rest}_F(x) &= \text{result}_F(x - \lceil x \rceil, \text{rnd}_F) \text{ if } x \in F \\
&= 0 \text{ if } x = -0 \\
&= \text{no_result}_F(x) \text{ otherwise}
\end{aligned}$$

5.2.5 Remainder after division with round to integer

$$\begin{aligned}
\text{residue}_F : F \times F &\rightarrow F \cup \{-0, \text{underflow}, \text{invalid}\} \\
\text{residue}_F(x, y) &= \text{result}_F(x - (\text{round}(x/y) \cdot y), \text{nearest}_F) \\
&\quad \text{if } x, y \in F \text{ and } y \neq 0 \text{ and} \\
&\quad \quad (x \geq 0 \text{ or } x - (\text{round}(x/y) \cdot y) \neq 0) \\
&= -0 \text{ if } x, y \in F \text{ and } y \neq 0 \text{ and} \\
&\quad \quad x < 0 \text{ and } x - (\text{round}(x/y) \cdot y) = 0 \\
&= -0 \text{ if } x = -0 \text{ and } y \in F \cup \{-\infty, +\infty\} \text{ and } y \neq 0 \\
&= x \text{ if } x \in F \text{ and } y \in \{-\infty, +\infty\} \\
&= \text{no_result}_F(x, y) \text{ otherwise}
\end{aligned}$$

5.2.6 Square root and reciprocal square root

$$\begin{aligned}
\text{sqrt}_F : F &\rightarrow F \cup \{\text{invalid}\} \\
\text{sqrt}_F(x) &= \text{nearest}_F(\sqrt{x}) \text{ if } x \in F \text{ and } x \geq 0 \\
&= x \text{ if } x \in \{-0, +\infty\} \\
&= \text{no_result}_F(x) \text{ otherwise}
\end{aligned}$$

$$\begin{aligned}
\text{rec_sqrt}_F : F &\rightarrow F \cup \{\text{infinitary}, \text{invalid}\} \\
\text{rec_sqrt}_F(x) &= \text{rnd}_F(1/\sqrt{x}) \text{ if } x \in F \text{ and } x > 0 \\
&= \text{infinitary}(+\infty) \text{ if } x \in \{-0, 0\} \\
&= 0 \text{ if } x = +\infty \\
&= \text{no_result}_F(x) \text{ otherwise}
\end{aligned}$$

5.2.7 Multiplication to higher precision floating point datatype

For the following operation, F' is a floating point datatype conforming to part 1, where $r_{F'} = r_F$ and $p_{F'} > p_F$.

$$\begin{aligned}
\text{mul}_{F \rightarrow F'} : F \times F &\rightarrow F' \cup \{-0, \text{underflow}, \text{overflow}\} \\
\text{mul}_{F \rightarrow F'}(x, y) &= \text{mul}_{F'}(\text{convert}_{F \rightarrow F'}(x), \text{convert}_{F \rightarrow F'}(y))
\end{aligned}$$

NOTES

- 1 $\text{convert}_{F \rightarrow F'}$ is specified in clause 5.4.4.
- 2 F' has the same radix as, but higher precision than F . If the precision is sufficiently much higher, rounding can be avoided. If also $\text{emin}_{F'}$ is sufficiently smaller than emin_F , underflow can be avoided, and if $\text{emax}_{F'}$ is sufficiently greater than emax_F , overflow can be avoided.

5.2.8 Support operations for extended floating point precision

These operations are useful when keeping guard digits or implementing extra precision floating point datatypes. The resulting datatypes, e.g. so-called doubled precision, do not necessarily conform to part 1.

$$\begin{aligned}
 add_lo_F : F \times F &\rightarrow F \cup \{\mathbf{underflow}\} \\
 add_lo_F(x, y) &= result_F((x + y) - rnd_F(x + y), rnd_F) \\
 &\quad \text{if } x, y \in F \\
 &= -0 && \text{if } x = -0 \text{ and } y \in F \cup \{-\infty, -0, +\infty\} \\
 &= -0 && \text{if } x \in F \cup \{-\infty, +\infty\} \text{ and } y = -0 \\
 &= y && \text{if } x = +\infty \text{ and } y \in F \cup \{+\infty\} \\
 &= y && \text{if } x = -\infty \text{ and } y \in F \cup \{-\infty\} \\
 &= x && \text{if } x \in F \text{ and } y \in \{-\infty, +\infty\} \\
 &= no_result2_F(x, y) && \text{otherwise}
 \end{aligned}$$

$$\begin{aligned}
 sub_lo_F : F \times F &\rightarrow F \cup \{\mathbf{underflow}\} \\
 sub_lo_F(x, y) &= add_lo_F(x, neg_F(y))
 \end{aligned}$$

NOTE 1 – If $rnd_style_F = nearest$, then, in the absence of notifications, add_lo_F and sub_lo_F return exact results.

$$\begin{aligned}
 mul_lo_F : F \times F &\rightarrow F \cup \{\mathbf{underflow}, \mathbf{overflow}\} \\
 mul_lo_F(x, y) &= result_F((x \cdot y) - rnd_F(x \cdot y), rnd_F) \\
 &\quad \text{if } x, y \in F \\
 &= mul_lo_F(0, y) && \text{if } x = -0 \text{ and } y \in F \cup \{-\infty, -0, +\infty\} \\
 &= mul_lo_F(x, 0) && \text{if } x \in F \cup \{-\infty, +\infty\} \text{ and } y = -0 \\
 &= mul_F(x, y) && \text{if } x \in \{-\infty, +\infty\} \text{ and } y \in F \cup \{-\infty, +\infty\} \\
 &= mul_F(x, y) && \text{if } x \in F \text{ and } y \in \{-\infty, +\infty\} \\
 &= no_result2_F(x, y) && \text{otherwise}
 \end{aligned}$$

NOTE 2 – In the absence of notifications, mul_lo_F returns an exact result.

$$\begin{aligned}
 div_rest_F : F \times F &\rightarrow F \cup \{\mathbf{underflow}, \mathbf{invalid}\} \\
 div_rest_F(x, y) &= result_F(x - (y \cdot rnd_F(x/y)), rnd_F) \\
 &\quad \text{if } x, y \in F \\
 &= div_rest_F(0, y) && \text{if } x = -0 \text{ and } y \in F \cup \{-\infty, -0, +\infty\} \\
 &= x && \text{if } x \in F \text{ and } y \in \{-\infty, +\infty\} \\
 &= x && \text{if } x \in \{-\infty, +\infty\} \text{ and } y \in F \\
 &= no_result2_F(x, y) && \text{otherwise}
 \end{aligned}$$

$$\begin{aligned}
 sqrt_rest_F : F &\rightarrow F \cup \{\mathbf{underflow}, \mathbf{invalid}\} \\
 sqrt_rest_F(x) &= result_F(x - (sqrt_F(x) \cdot sqrt_F(x)), rnd_F) \\
 &\quad \text{if } x \in F \text{ and } x \geq 0 \\
 &= -0 && \text{if } x = -0
 \end{aligned}$$

$$\begin{aligned}
&= +\infty && \text{if } x = +\infty \\
&= no_result_F(x) && \text{otherwise}
\end{aligned}$$

NOTE 3 – $sqr_rest_F(x)$ is exact when there is no **underflow**.

5.3 Elementary transcendental floating point operations

The specifications for each of the floating point transcendental operations and the floating point conversion operations (clause 5.4) use an approximation helper function. The approximation helper functions are ideally identical to the true mathematical functions. However, that would imply a maximum error for the corresponding operation of 0.5 ulp (i.e., the minimum value for operations that are not always exact). This part does not require that the maximum error is only 0.5 ulp for the operations specified in clause 5.3, but allows the maximum error to be a bit bigger. To express this, the approximation helper functions need not be identical to the mathematical elementary transcendental functions, but are allowed to be approximate. The approximation helper functions shall be defined for the elements of its given argument signature where the corresponding mathematical function is also defined, unless otherwise noted. The requirements on approximation helper functions apply only where the approximation helper functions are defined.

5.3.1 Maximum error requirements

The approximation helper functions for the individual operations in these subclauses have maximum error parameters that describe the maximum *relative* error, in ulps, of the helper function composed with $nearest_F$, for non-subnormal and non-zero results. The maximum error parameters also describe the maximum *absolute* error, in ulps, for $-fminN_F$, $fminN_F$, subnormal, or zero results and underflow continuation values if $denorm_F = \mathbf{true}$. All maximum error parameters shall have a value that is ≥ 0.5 . For the maximum value for the maximum error parameters, see the specification of each of the maximum error parameters. See also Annex A, on partial conformity. The relevant maximum error parameters shall be made available to programs.

When the maximum error for an approximation helper function h_F , approximating f , is $max_error_op_F$, then for all arguments $x, \dots \in F \times \dots$ the following equation shall hold:

$$|f(x, \dots) - nearest_F(h_F(x, \dots))| \leq max_error_op_F \cdot r_F^{e_F(f(x, \dots)) - p_F}$$

NOTES

- 1 Partially conforming implementations may have greater values for maximum error parameters than stipulated below. See annex A.
- 2 For most positive (and not too small) return values t , the true result is thus claimed to be in the interval $[t - (max_error_op_F \cdot ulp_F(t)), t + (max_error_op_F \cdot ulp_F(t))]$. But if the return value is exactly r_F^n for some not too small $n \in \mathbb{Z}$, then the true result is claimed to be in the interval $[t - (max_error_op_F \cdot ulp_F(t)/r_F), t + (max_error_op_F \cdot ulp_F(t))]$. Similarly for negative return values.

The results of the approximating helper functions in this clause must be exact for certain arguments as detailed below, and may be exact for all arguments. If the approximating helper function is exact for all arguments, then the corresponding maximum error parameter should have the value 0.5, the minimum value.

5.3.2 Sign requirements

For this part, the approximation helper functions shall be zero exactly at the points where the approximated mathematical function is exactly zero. For this part, at points where the approximation helper functions are not zero, they shall have the same sign as the approximated mathematical function at that point. For the radian trigonometric helper functions, these zero and sign requirements are imposed only for arguments, x , such that $|x| \leq \text{big_angle_r}_F$ (see clause 5.3.8).

NOTE – For the operations, the continuation value after an **underflow** may be zero (including negative zero) as given by result_F^* (see below), even though the approximation helper function is not zero at that point. Such zero results are required to be accompanied by an **underflow** notification. When appropriate, zero may also be returned for IEC 60559 infinities arguments. See the individual specifications.

5.3.3 Monotonicity requirements

For this part, each approximation helper function shall be a monotonic approximation to the mathematical function it is approximating, except:

- a) For the radian trigonometric approximation helper functions, the monotonic approximation requirement is imposed only for arguments, x , such that $|x| \leq \text{big_angle_r}_F$ (see clause 5.3.8).
- b) The argument angular unit trigonometric and argument angular unit inverse trigonometric approximating helper functions, as well as the angular unit conversion helper functions, are excepted from the monotonic approximation requirement for the angular unit argument(s).

5.3.4 The result^* helper function

The result_F^* helper function is similar to the result_F helper function (see clause 5.2.1), but is simplified compared to result_F concerning **underflow**: result_F^* always underflows for non-zero arguments that have an absolute value less than $\text{fmin}N_F - (\text{fmin}D_F/r_F)$, whereas result_F does not necessarily underflow in that case. This difference from result_F is made since the argument to result_F^* might not be exact. To return **underflow** or not, for a tiny result, based upon an inexact argument would be misleading. For the operations specified using result_F^* where the specification implies that there can never be any denormalisation loss for certain tiny results, **underflow** is instead explicitly avoided.

$$\begin{aligned} \text{result}_F^* : \mathcal{R} \times (\mathcal{R} \rightarrow F^*) &\rightarrow F \cup \{\text{underflow}, \text{overflow}\} \\ \text{result}_F^*(x, \text{rnd}) &= \text{underflow}(c) && \text{if } x \in \mathcal{R} \text{ and } \text{denorm}_F = \text{true} \text{ and} \\ & && |\text{rnd}(x)| < \text{fmin}N_F \text{ and } x \neq 0 \\ &= \text{result}_F(x, \text{rnd}) && \text{otherwise} \end{aligned}$$

where

$$\begin{aligned} c &= \text{rnd}(x) && \text{when } \text{rnd}(x) \neq 0 \text{ or } x > 0, \\ c &= -0 && \text{when } \text{rnd}(x) = 0 \text{ and } x < 0 \end{aligned}$$

5.3.5 Hypotenuse

There shall be a maximum error parameter for the hypot_F operation:

$$\text{max_error_hypot}_F \in F$$

The max_error_hypot_F parameter shall have a value that is ≤ 1 .

The hypot_F^* approximation helper function:

$$\text{hypot}_F^* : F \times F \rightarrow \mathcal{R}$$

$\text{hypot}_F^*(x, y)$ returns a close approximation to $\sqrt{x^2 + y^2}$ in $\mathcal{R}$, with maximum error max_error_hypot_F .

Further requirements on the hypot_F^* approximation helper function are:

$$\begin{array}{ll} \text{hypot}_F^*(x, y) = \text{hypot}_F^*(y, x) & \text{if } x, y \in F \\ \text{hypot}_F^*(-x, y) = \text{hypot}_F^*(x, y) & \text{if } x, y \in F \\ \text{hypot}_F^*(x, y) \geq \max\{|x|, |y|\} & \text{if } x, y \in F \\ \text{hypot}_F^*(x, y) \leq |x| + |y| & \text{if } x, y \in F \\ \text{hypot}_F^*(x, y) \geq 1 & \text{if } x, y \in F \text{ and } \sqrt{x^2 + y^2} \geq 1 \\ \text{hypot}_F^*(x, y) \leq 1 & \text{if } x, y \in F \text{ and } \sqrt{x^2 + y^2} \leq 1 \end{array}$$

The hypot_F operation:

$$\begin{array}{ll} \text{hypot}_F : F \times F \rightarrow F \cup \{\text{underflow}, \text{overflow}\} \\ \text{hypot}_F(x, y) & = \text{result}_F^*(\text{hypot}_F^*(x, y), \text{nearest}_F) \\ & \text{if } x, y \in F \\ & = \text{hypot}_F(0, y) & \text{if } x = -0 \text{ and } y \in F \cup \{-\infty, -0, +\infty\} \\ & = \text{hypot}_F(x, 0) & \text{if } y = -0 \text{ and } x \in F \cup \{-\infty, +\infty\} \\ & = +\infty & \text{if } x \in \{-\infty, +\infty\} \text{ and } y \in F \cup \{-\infty, +\infty\} \\ & = +\infty & \text{if } y \in \{-\infty, +\infty\} \text{ and } x \in F \\ & = \text{no_result}_F(x, y) & \text{otherwise} \end{array}$$

5.3.6 Operations for exponentiations and logarithms

There shall be two maximum error parameters for approximate exponentiations and logarithms:

$$\text{max_error_exp}_F \in F$$

$$\text{max_error_power}_F \in F$$

The max_error_exp_F parameter shall have a value that is $\leq 1.5 \cdot \text{rnd_error}_F$. The max_error_power_F parameter shall have a value that is $\leq 2 \cdot \text{rnd_error}_F$.

5.3.6.1 Integer power of argument base

The $\text{power}_{F,I}^*$ approximation helper function:

$$\text{power}_{F,I}^* : F \times I \rightarrow \mathcal{R}$$

$\text{power}_{F,I}^*(x, y)$ returns a close approximation to x^y in $\mathcal{R}$, with maximum error max_error_power_F .

Further requirements on the $\text{power}_{F,I}^*$ approximation helper function are:

$power_{F,I}^*(x, y) = x^y$	if $x \in \mathcal{Z} \cap F$ and $y \in I$ and ($ x = 1$ or $y > 0$)
$power_{F,I}^*(x, 1) = x$	if $x \in F$
$power_{F,I}^*(x, 0) = 1$	if $x \in F$ and $x \neq 0$
$power_{F,I}^*(x, y) < fminD_F/2$	if $x \in F$ and $x > 0$ and $y \in I$ and $x^y < fminD_F/3$
$power_{F,I}^*(x, y) = power_{F,I}^*(-x, y)$	if $x \in F$ and $x < 0$ and $y \in I$ and $2 y$
$power_{F,I}^*(x, y) = -power_{F,I}^*(-x, y)$	if $x \in F$ and $x < 0$ and $y \in I$ and not $2 y$

The relationship to other $power_{FI'}$ helper functions for any $power_{FI'}$ operations in the same library shall be:

$$power_{FI}^*(x, y) = power_{FI'}^*(x, y) \quad \text{if } x \in F \text{ and } y \in I \cap I'$$

The $power_{FI}$ operation:

$$power_{FI} : F \times I \rightarrow F \cup \{\text{underflow}, \text{overflow}, \text{infinitary}\}$$

$$\begin{aligned}
 power_{FI}(x, y) &= result_F^*(power_{FI}^*(x, y), nearest_F) && \text{if } x \in F \text{ and } x \neq 0 \text{ and } y \in I \\
 &= +\infty && \text{if } x = -\infty \text{ and } y \in I \text{ and } y > 0 \text{ and } 2|y \\
 &= -\infty && \text{if } x = -\infty \text{ and } y \in I \text{ and } y > 0 \text{ and not } 2|y \\
 &= 0 && \text{if } x = -0 \text{ and } y \in I \text{ and } y > 0 \text{ and } 2|y \\
 &= -0 && \text{if } x = -0 \text{ and } y \in I \text{ and } y > 0 \text{ and not } 2|y \\
 &= 0 && \text{if } x = 0 \text{ and } y \in I \text{ and } y > 0 \\
 &= +\infty && \text{if } x = +\infty \text{ and } y \in I \text{ and } y > 0 \\
 &= 1 && \text{if } x \in \{-\infty, -0, 0, +\infty\} \text{ and } y = 0 \\
 &= 0 && \text{if } x = -\infty \text{ and } y \in I \text{ and } y < 0 \text{ and } 2|y \\
 &= -0 && \text{if } x = -\infty \text{ and } y \in I \text{ and } y < 0 \text{ and not } 2|y \\
 &= \text{infinitary}(+\infty) && \text{if } x = -0 \text{ and } y \in I \text{ and } y < 0 \text{ and } 2|y \\
 &= \text{infinitary}(-\infty) && \text{if } x = -0 \text{ and } y \in I \text{ and } y < 0 \text{ and not } 2|y \\
 &= \text{infinitary}(+\infty) && \text{if } x = 0 \text{ and } y \in I \text{ and } y < 0 \\
 &= 0 && \text{if } x = +\infty \text{ and } y \in I \text{ and } y < 0 \\
 &= no_result_F(x) && \text{otherwise}
 \end{aligned}$$

NOTES

- 1 $power_{FI}(x, y)$ will overflow approximately when $x^y > fmax_F$, i.e., if $x > 1$, approximately when $y > \log_x(fmax_F)$, and if $0 < x < 1$, approximately when $y < \log_x(fmax_F)$ (which is then negative). It will not overflow when $x = 0$ or when $x = 1$.
- 2 $power_I$ (in clause 5.1.4) does not allow most negative *exponents* (unless $|x| = 1$) since the exact result then is not in $\mathcal{Z}$ unless $|x| = 1$. $power_F$ (in clause 5.3.6.6) does not allow any negative *bases* since the (exact) result is not in $\mathcal{R}$ unless the exponent is integer. $power_{FI}$ takes care of this latter case, where all exponents are ensured to be integers that have not arisen from implicit floating point rounding.

5.3.6.2 Natural exponentiation

The exp_F^* approximation helper function:

$$exp_F^* : F \rightarrow \mathcal{R}$$

5.3.6 Operations for exponentiations and logarithms

$\exp_F^*(x)$ returns a close approximation to e^x in $\mathcal{R}$, with maximum error $\max_error_exp_F$.

Further requirements on the $\exp_F^*$ approximation helper function are:

$$\begin{aligned} \exp_F^*(1) &= e \\ \exp_F^*(x) &= 1 && \text{if } x \in F \text{ and } \exp_F^*(x) \neq e^x \text{ and} \\ & && \ln(1 - (\epsilon_F/(2 \cdot r_F))) < x \text{ and} \\ & && x < \ln(1 + (\epsilon_F/2)) \\ \exp_F^*(x) &< fminD_F/2 && \text{if } x \in F \text{ and } x < \ln(fminD_F) - 3 \end{aligned}$$

The $\exp_F$ operation:

$$\begin{aligned} \exp_F : F &\rightarrow F \cup \{\text{underflow}, \text{overflow}\} \\ \exp_F(x) &= result_F^*(\exp_F^*(x), nearest_F) \\ &= 1 && \text{if } x \in F \\ &= +\infty && \text{if } x = -0 \\ &= 0 && \text{if } x = +\infty \\ &= 0 && \text{if } x = -\infty \\ &= no_result_F(x) && \text{otherwise} \end{aligned}$$

NOTES

- 1 $\exp_F(1) = nearest_F(e)$.
- 2 $\exp_F(x)$ will overflow approximately when $x > \ln(fmax_F)$.

5.3.6.3 Natural exponentiation, minus one

The $\expm1_F^*$ approximation helper function:

$$\expm1_F^* : F \rightarrow \mathcal{R}$$

$\expm1_F^*(x)$ returns a close approximation to $e^x - 1$ in $\mathcal{R}$, with maximum error $\max_error_exp_F$.

Further requirements on the $\expm1_F^*$ approximation helper function are:

$$\begin{aligned} \expm1_F^*(1) &= e - 1 \\ \expm1_F^*(x) &= x && \text{if } x \in F \text{ and } \expm1_F^*(x) \neq e^x - 1 \text{ and} \\ & && -\epsilon_F/r_F \leq x < 0.5 \cdot \epsilon_F/r_F \\ \expm1_F^*(x) &= -1 && \text{if } x \in F \text{ and } \expm1_F^*(x) \neq e^x - 1 \text{ and} \\ & && x < \ln(\epsilon_F/(3 \cdot r_F)) \end{aligned}$$

The relationship to the $\exp_F^*$ approximation helper function for the $\exp_F$ operation in the same library shall be:

$$\expm1_F^*(x) \leq \exp_F^*(x) \quad \text{if } x \in F$$

The $\expm1_F$ operation:

$$\begin{aligned} \expm1_F : F &\rightarrow F \cup \{\text{overflow}\} \\ \expm1_F(x) &= result_F^*(\expm1_F^*(x), nearest_F) \\ &= x && \text{if } x \in F \text{ and } |x| \geq fminN_F \\ &= -0 && \text{if } x \in F \text{ and } |x| < fminN_F \\ &= +\infty && \text{if } x = -0 \\ &= -1 && \text{if } x = +\infty \\ &= -1 && \text{if } x = -\infty \\ &= no_result_F(x) && \text{otherwise} \end{aligned}$$

NOTES

- 1 **underflow** is explicitly avoided. Part 1 requires that $fminN_F \leq \epsilon_F$. This part requires that $fminN_F < 0.5 \cdot \epsilon_F / r_F$, so that underflow can be avoided here.
- 2 $expm1_F(1) = nearest_F(e - 1)$.
- 3 $expm1_F(x)$ will overflow approximately when $x > \ln(fmax_F)$.

5.3.6.4 Exponentiation of 2

The $exp2_F^*$ approximation helper function:

$$exp2_F^* : F \rightarrow \mathcal{R}$$

$exp2_F^*(x)$ returns a close approximation to 2^x in $\mathcal{R}$, with maximum error $max_error_exp_F$.

Further requirements on the $exp2_F^*$ approximation helper function are:

$$\begin{aligned}
 exp2_F^*(x) &= 1 && \text{if } x \in F \text{ and } exp2_F^*(x) \neq 2^x \text{ and} \\
 & && \log_2(1 - (\epsilon_F / (2 \cdot r_F))) < x \text{ and} \\
 & && x < \log_2(1 + (\epsilon_F / 2)) \\
 exp2_F^*(x) &= 2^x && \text{if } x \in F \cap \mathcal{Z} \text{ and } 2^x \in F \\
 exp2_F^*(x) &< fminD_F / 2 && \text{if } x \in F \text{ and } x < \log_2(fminD_F) - 3
 \end{aligned}$$

The $exp2_F$ operation:

$$\begin{aligned}
 exp2_F : F &\rightarrow F \cup \{\mathbf{underflow}, \mathbf{overflow}\} \\
 exp2_F(x) &= result_F^*(exp2_F^*(x), nearest_F) \\
 &= 1 && \text{if } x \in F \\
 &= +\infty && \text{if } x = -0 \\
 &= 0 && \text{if } x = +\infty \\
 &= 0 && \text{if } x = -\infty \\
 &= no_result_F(x) && \text{otherwise}
 \end{aligned}$$

NOTE – $exp2_F(x)$ will overflow approximately when $x > \log_2(fmax_F)$.

5.3.6.5 Exponentiation of 10

The $exp10_F^*$ approximation helper function:

$$exp10_F^* : F \rightarrow \mathcal{R}$$

$exp10_F^*(x)$ returns a close approximation to 10^x in $\mathcal{R}$, with maximum error $max_error_exp_F$.

Further requirements on the $exp10_F^*$ approximation helper function are:

$$\begin{aligned}
 exp10_F^*(x) &= 1 && \text{if } x \in F \text{ and } exp10_F^*(x) \neq 10^x \text{ and} \\
 & && \log_{10}(1 - (\epsilon_F / (2 \cdot r_F))) < x \text{ and} \\
 & && x < \log_{10}(1 + (\epsilon_F / 2)) \\
 exp10_F^*(x) &= 10^x && \text{if } x \in F \cap \mathcal{Z} \text{ and } 10^x \in F \\
 exp10_F^*(x) &< fminD_F / 2 && \text{if } x \in F \text{ and } x < \log_{10}(fminD_F) - 3
 \end{aligned}$$

The $exp10_F$ operation:

$$exp10_F : F \rightarrow F \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$

$$\begin{aligned}
exp10_F(x) &= result_F^*(exp10_F^*(x), nearest_F) && \text{if } x \in F \\
&= 1 && \text{if } x = -0 \\
&= +\infty && \text{if } x = +\infty \\
&= 0 && \text{if } x = -\infty \\
&= no_result_F(x) && \text{otherwise}
\end{aligned}$$

NOTE – $exp10_F(x)$ will overflow approximately when $x > \log_{10}(fmax_F)$.

5.3.6.6 Exponentiation of argument base

The $power_F^*$ approximation helper function:

$$power_F^* : F \times F \rightarrow \mathcal{R}$$

$power_F^*(x, y)$ returns a close approximation to x^y in $\mathcal{R}$, with maximum error $max_error_power_F$.

The $power_F^*$ helper function need be defined only for first arguments that are greater than 0.

Further requirements on the $power_F^*$ approximation helper function are:

$$\begin{aligned}
power_F^*(1, y) &= 1 && \text{if } y \in F \\
power_F^*(x, 0) &= 1 && \text{if } x \in F \text{ and } x > 0 \\
power_F^*(x, 1) &= x && \text{if } x \in F \text{ and } x > 0 \\
power_F^*(x, y) &< fminD_F/2 && \text{if } x \in F \text{ and } x > 0 \text{ and } y \in F \text{ and } x^y < fminD_F/3
\end{aligned}$$

The relationship to the $power_{FI}^*$ approximation helper functions for any $power_{FI}$ operations in the same library shall be:

$$power_F^*(x, y) = power_{F,I}^*(x, y) \quad \text{if } x \in F \text{ and } x > 0 \text{ and } y \in I \cap F$$

The $power_F$ operation:

$$\begin{aligned}
power_F : F \times F &\rightarrow F \cup \{\text{underflow}, \text{overflow}, \text{infinitary}, \text{invalid}\} \\
power_F(x, y) &= result_F^*(power_F^*(x, y), nearest_F) && \text{if } x \in F \text{ and } x > 0 \text{ and } y \in F \\
&= power_F(0, y) && \text{if } x = -0 \text{ and } y \in F \cup \{-\infty, -0, +\infty\} \\
&= power_F(x, 0) && \text{if } y = -0 \text{ and } x \in F \cup \{-\infty, +\infty\} \\
&= +\infty && \text{if } x = +\infty \text{ and } ((y \in F \text{ and } y > 0) \text{ or } y = +\infty) \\
&= +\infty && \text{if } x \in F \text{ and } x > 1 \text{ and } y = +\infty \\
&= 0 && \text{if } x \in F \text{ and } 0 \leq x < 1 \text{ and } y = +\infty \\
&= 0 && \text{if } x = 0 \text{ and } y \in F \text{ and } y > 0 \\
&= \text{infinitary}(+\infty) && \text{if } x = 0 \text{ and } y \in F \text{ and } y < 0 \\
&= +\infty && \text{if } x \in F \text{ and } 0 \leq x < 1 \text{ and } y = -\infty \\
&= 0 && \text{if } x \in F \text{ and } x > 1 \text{ and } y = -\infty \\
&= 0 && \text{if } x = +\infty \text{ and } ((y \in F \text{ and } y < 0) \text{ or } y = -\infty) \\
&= no_result2_F(x, y) && \text{otherwise}
\end{aligned}$$

NOTE – $power_F(x, y)$ will overflow approximately when $x^y > fmax_F$, i.e., if $x > 1$, approximately when $y > \log_x(fmax_F)$, and if $0 < x < 1$, approximately when $y < \log_x(fmax_F)$ (which is a negative number). It will not overflow when $x = 0$ or when $x = 1$.

5.3.6.7 Exponentiation of one plus the argument base, minus one

The $power1pm1_F^*$ approximation helper function:

$$power1pm1_F^* : F \times F \rightarrow \mathcal{R}$$

$power1pm1_F^*(x, y)$ returns a close approximation to $(1 + x)^y - 1$ in $\mathcal{R}$, with maximum error $max_error_power_F$. The $power1pm1_F^*$ helper function need be defined only for first arguments that are greater than or equal to -1 .

Further requirements on the $power1pm1_F^*$ approximation helper function are:

$$\begin{aligned} power1pm1_F^*(x, y) &= (1 + x)^y - 1 && \text{if } x, y \in F \cap \mathcal{Z} \text{ and } x \geq -1 \text{ and } y > 0 \\ power1pm1_F^*(x, 1) &= x && \text{if } x, 1 + x \in F \text{ and } x > -1 \\ power1pm1_F^*(-1, y) &= -1 && \text{if } y \in F \text{ and } y > 0 \\ power1pm1_F^*(x, y) &= -1 && \text{if } x \in F \text{ and } x > -1 \text{ and } y \in F \text{ and} \\ &&& power1pm1_F^*(x, y) \neq (1 + x)^y - 1 \text{ and} \\ &&& (1 + x)^y < epsilon_F / (3 \cdot r_F) \end{aligned}$$

The relationship to the $power_F^*$ approximation helper function for the $power_F$ operation in the same library shall be:

$$power1pm1_F^*(x, y) \leq power_F^*(1 + x, y) \quad \text{if } x, 1 + x \in F \text{ and } x > -1 \text{ and } y \in F$$

NOTE 1 – $power1pm1_F^*(x, y) \approx y \cdot \ln(1 + x)$ if $x \in F$ and $x > -1$ and $y \in F$ and $|y \cdot \ln(1 + x)| < epsilon_F / r_F$.

The $power1pm1_F$ operation:

$$\begin{aligned} power1pm1_F : F \times F &\rightarrow F \cup \{-0, \text{underflow}, \text{overflow}, \text{infinitary}, \text{invalid}\} \\ power1pm1_F(x, y) &= result_F^*(power1pm1_F^*(x, y), nearest_F) \\ &= mul_F(x, y) && \text{if } x \in F \text{ and } x > -1 \text{ and } x \neq 0 \text{ and } y \in F \text{ and } y \neq 0 \\ &= mul_F(x, y) && \text{if } x \in \{-0, 0\} \text{ and } y \in F \text{ and } y \neq 0 \\ &= mul_F(x, y) && \text{if } y \in \{-0, 0\} \text{ and } x \in F \text{ and } x > -1 \\ &= +\infty && \text{if } x = +\infty \text{ and } ((y \in F \text{ and } y > 0) \text{ or } y = +\infty) \\ &= +\infty && \text{if } x \in F \text{ and } x > 0 \text{ and } y = +\infty \\ &= -1 && \text{if } x \in F \text{ and } -1 \leq x < 0 \text{ and } y = +\infty \\ &= -1 && \text{if } x = -1 \text{ and } y \in F \text{ and } y > 0 \\ &= \text{infinitary}(+\infty) && \text{if } x = -1 \text{ and } y \in F \text{ and } y < 0 \\ &= +\infty && \text{if } x \in F \text{ and } -1 \leq x < 0 \text{ and } y = -\infty \\ &= -1 && \text{if } x \in F \text{ and } x > 0 \text{ and } y = -\infty \\ &= -1 && \text{if } x = +\infty \text{ and } ((y \in F \text{ and } y < 0) \text{ or } y = -\infty) \\ &= no_result2_F(x, y) && \text{otherwise} \end{aligned}$$

NOTE 2 – $power1pm1_F(x, y)$ will overflow approximately when $(1 + x)^y > fmax_F$, i.e., if $x > 0$, approximately when $y > \log_{1+x}(fmax_F)$, and if $-1 < x < 0$, approximately when $y < \log_{1+x}(fmax_F)$. It will not overflow when $x \in \{-1, 0\}$.

5.3.6.8 Natural logarithm

The ln_F^* approximation helper function:

$$ln_F^* : F \cup \{e\} \rightarrow \mathcal{R}$$

$\ln_F^*(x)$ returns a close approximation to $\ln(x)$ in $\mathcal{R}$, with maximum error $\max_error_exp_F$.

A further requirement on the $\ln_F^*$ approximation helper function is:

$$\ln_F^*(e) = 1$$

The $\ln_F$ operation:

$$\begin{aligned} \ln_F : F &\rightarrow F \cup \{\mathbf{infinitary}, \mathbf{invalid}\} \\ \ln_F(x) &= \text{result}_F^*(\ln_F^*(x), \text{nearest}_F) \\ &\quad \text{if } x \in F \text{ and } x > 0 \\ &= \mathbf{infinitary}(-\infty) && \text{if } x \in \{-0, 0\} \\ &= +\infty && \text{if } x = +\infty \\ &= \text{no_result}_F(x) && \text{otherwise} \end{aligned}$$

5.3.6.9 Natural logarithm of one plus the argument

The $\ln1p_F^*$ approximation helper function:

$$\ln1p_F^* : F \cup \{e - 1\} \rightarrow \mathcal{R}$$

$\ln1p_F^*(x)$ returns a close approximation to $\ln(1 + x)$ in $\mathcal{R}$, with maximum error $\max_error_exp_F$.

Further requirements on the $\ln1p_F^*$ approximation helper function are:

$$\begin{aligned} \ln1p_F^*(e - 1) &= 1 \\ \ln1p_F^*(x) &= x && \text{if } x \in F \text{ and } \ln1p_F^*(x) \neq \ln(1 + x) \text{ and} \\ &&& -0.5 \cdot \text{epsilon}_F / r_F < x \leq \text{epsilon}_F / r_F \end{aligned}$$

The relationship to the $\ln_F^*$ approximation helper function for the $\ln_F$ operation in the same library shall be:

$$\ln1p_F^*(x) \geq \ln_F^*(x) \quad \text{if } x \in F \text{ and } x > 0$$

The $\ln1p_F$ operation:

$$\begin{aligned} \ln1p_F : F &\rightarrow F \cup \{\mathbf{infinitary}, \mathbf{invalid}\} \\ \ln1p_F(x) &= \text{result}_F^*(\ln1p_F^*(x), \text{nearest}_F) \\ &\quad \text{if } x \in F \text{ and } x > -1 \text{ and } |x| \geq \text{fmin}N_F \\ &= x && \text{if } x \in F \text{ and } |x| < \text{fmin}N_F \\ &= -0 && \text{if } x = -0 \\ &= \mathbf{infinitary}(-\infty) && \text{if } x = -1 \\ &= +\infty && \text{if } x = +\infty \\ &= \text{no_result}_F(x) && \text{otherwise} \end{aligned}$$

NOTE – **underflow** is explicitly avoided. Part 1 requires that $\text{fmin}N_F \leq \text{epsilon}_F$. This part requires that $\text{fmin}N_F < 0.5 \cdot \text{epsilon}_F / r_F$, so that underflow can be avoided here.

5.3.6.10 2-logarithm

The $\log2_F^*$ approximation helper function:

$$\log2_F^* : F \rightarrow \mathcal{R}$$

$\log2_F^*(x)$ returns a close approximation to $\log_2(x)$ in $\mathcal{R}$, with maximum error $\max_error_exp_F$.

A further requirement on the $\log2_F^*$ approximation helper function is:

$$\log 2_F^*(x) = \log_2(x) \quad \text{if } x \in F \text{ and } \log_2(x) \in \mathcal{Z}$$

The $\log 2_F$ operation:

$$\begin{aligned} \log 2_F : F &\rightarrow F \cup \{\mathbf{infinitary}, \mathbf{invalid}\} \\ \log 2_F(x) &= \mathit{result}_F^*(\log 2_F^*(x), \mathit{nearest}_F) \\ &\quad \text{if } x \in F \text{ and } x > 0 \\ &= \mathbf{infinitary}(-\infty) && \text{if } x \in \{-0, 0\} \\ &= +\infty && \text{if } x = +\infty \\ &= \mathit{no_result}_F(x) && \text{otherwise} \end{aligned}$$

5.3.6.11 10-logarithm

The $\log 10_F^*$ approximation helper function:

$$\log 10_F^* : F \rightarrow \mathcal{R}$$

$\log 10_F^*(x)$ returns a close approximation to $\log_{10}(x)$ in $\mathcal{R}$, with maximum error $\mathit{max_error_exp}_F$.

A further requirement on the $\log 10_F^*$ approximation helper function is:

$$\log 10_F^*(x) = \log_{10}(x) \quad \text{if } x \in F \text{ and } \log_{10}(x) \in \mathcal{Z}$$

The $\log 10_F$ operation:

$$\begin{aligned} \log 10_F : F &\rightarrow F \cup \{\mathbf{infinitary}, \mathbf{invalid}\} \\ \log 10_F(x) &= \mathit{result}_F^*(\log 10_F^*(x), \mathit{nearest}_F) \\ &\quad \text{if } x \in F \text{ and } x > 0 \\ &= \mathbf{infinitary}(-\infty) && \text{if } x \in \{-0, 0\} \\ &= +\infty && \text{if } x = +\infty \\ &= \mathit{no_result}_F(x) && \text{otherwise} \end{aligned}$$

5.3.6.12 Argument base logarithm

The $\log base_F^*$ approximation helper function:

$$\log base_F^* : F \times F \rightarrow \mathcal{R}$$

$\log base_F^*(x, y)$ returns a close approximation to $\log_x(y)$ in $\mathcal{R}$, with maximum error $\mathit{max_error_power}_F$.

A further requirement on the $\log base_F^*$ approximation helper function is:

$$\log base_F^*(x, x) = 1 \quad \text{if } x \in F \text{ and } x > 0 \text{ and } x \neq 1$$

The $\log base_F$ operation:

$$\begin{aligned} \log base_F : F \times F &\rightarrow F \cup \{-0, \mathbf{infinitary}, \mathbf{invalid}\} \\ \log base_F(x, y) &= \mathit{result}_F^*(\log base_F^*(x, y), \mathit{nearest}_F) \\ &\quad \text{if } x \in F \text{ and } x > 0 \text{ and } x \neq 1 \text{ and } y \in F \text{ and } y > 0 \\ &= \log base_F(0, y) && \text{if } x = -0 \text{ and } y \in F \cup \{-\infty, -0, +\infty\} \\ &= \log base_F(x, 0) && \text{if } y = -0 \text{ and } x \in F \cup \{-\infty, +\infty\} \\ &= \mathbf{infinitary}(+\infty) && \text{if } x = 1 \text{ and } y \in F \text{ and } y > 1 \\ &= \mathbf{infinitary}(-\infty) && \text{if } x = 1 \text{ and } y \in F \text{ and } 0 \leq y < 1 \\ &= 0 && \text{if } x = +\infty \text{ and } y \in F \text{ and } y \geq 1 \\ &= +\infty && \text{if } x \in F \text{ and } 1 \leq x \text{ and } y = +\infty \end{aligned}$$

$= -\infty$	if $x \in F$ and $0 < x < 1$ and $y = +\infty$
$= -0$	if $x = 0$ and $y \in F$ and $y \geq 1$
$= 0$	if $x = 0$ and $y \in F$ and $0 < y < 1$
$= \text{infinitary}(+\infty)$	if $x \in F$ and $0 < x < 1$ and $y = 0$
$= \text{infinitary}(-\infty)$	if $x \in F$ and $1 < x$ and $y = 0$
$= -0$	if $x = +\infty$ and $y \in F$ and $0 < y < 1$
$= \text{no_result}_{2F}(x, y)$	otherwise

5.3.6.13 Argument base logarithm of one plus each argument

The logbase1p1p_F^* approximation helper function:

$$\text{logbase1p1p}_F^* : F \times F \rightarrow \mathcal{R}$$

$\text{logbase1p1p}_F^*(x, y)$ returns a close approximation to $\log_{(1+x)}(1 + y)$ in $\mathcal{R}$, with maximum error max_error_power_F .

A further requirements on logbase1p1p_F^* approximation helper function is:

$$\text{logbase1p1p}_F^*(x, x) = 1 \quad \text{if } x \in F \text{ and } x > -1 \text{ and } x \neq 0$$

The logbase1p1p_F operation:

$$\text{logbase1p1p}_F : F \times F \rightarrow F \cup \{-0, \text{underflow}, \text{infinitary}, \text{invalid}\}$$

$$\begin{aligned} \text{logbase1p1p}_F(x, y) &= \text{result}_F^*(\text{logbase1p1p}_F^*(x, y), \text{nearest}_F) \\ &\quad \text{if } x \in F \text{ and } x > -1 \text{ and } x \neq 0 \text{ and } \\ &\quad \quad y \in F \text{ and } y > -1 \text{ and } y \neq 0 \\ &= \text{div}_F(y, x) \quad \text{if } x \in \{-0, 0\} \text{ and } \\ &\quad ((y \in F \text{ and } y > -1 \text{ and } y \neq 0) \text{ or } y = +\infty) \\ &= \text{div}_F(y, x) \quad \text{if } y \in \{-0, 0\} \text{ and } \\ &\quad ((x \in F \text{ and } x > -1) \text{ or } x = +\infty) \\ &= 0 \quad \text{if } x = +\infty \text{ and } y \in F \text{ and } y \geq 0 \\ &= +\infty \quad \text{if } x \in F \text{ and } 0 < x \text{ and } y = +\infty \\ &= -\infty \quad \text{if } x \in F \text{ and } -1 < x < 0 \text{ and } y = +\infty \\ &= -0 \quad \text{if } x = -1 \text{ and } y \in F \text{ and } y \geq 0 \\ &= 0 \quad \text{if } x = -1 \text{ and } y \in F \text{ and } -1 < y < 0 \\ &= \text{infinitary}(+\infty) \quad \text{if } x \in F \text{ and } -1 < x < 0 \text{ and } y = -1 \\ &= \text{infinitary}(-\infty) \quad \text{if } x \in F \text{ and } 0 < x \text{ and } y = -1 \\ &= -0 \quad \text{if } x = +\infty \text{ and } y \in F \text{ and } -1 < y < 0 \\ &= \text{no_result}_{2F}(x, y) \quad \text{otherwise} \end{aligned}$$

5.3.7 Introduction to operations for trigonometric elementary functions

Two different operations for each of sin, cos, tan, cot, sec, csc, arcsin, arccos, arctan, arccot, arccotc, arcsec, and arccsc are specified. One version for radians and one version where the angular unit is given as a parameter.

For use in the specifications below, define the following mathematical functions:

$$\begin{aligned}
rad &: \mathcal{R} \rightarrow \mathcal{R} \\
axis_rad &: \mathcal{R} \rightarrow \{(1, 0), (0, 1), (-1, 0), (0, -1)\} \times \mathcal{R} \\
arc &: \mathcal{R} \times \mathcal{R} \rightarrow \mathcal{R}
\end{aligned}$$

The *rad*, angular value normalisation, function is defined by

$$rad(x) = x - \text{round}(x/(2 \cdot \pi)) \cdot 2 \cdot \pi$$

The *axis_rad* function is defined by

$$\begin{aligned}
axis_rad(x) &= ((1, 0), \arcsin(\sin(x))) && \text{if } \cos(x) \geq 1/\sqrt{2} \\
&= ((0, 1), \arcsin(\cos(x))) && \text{if } \sin(x) > 1/\sqrt{2} \\
&= ((-1, 0), \arcsin(\sin(x))) && \text{if } \cos(x) \leq -1/\sqrt{2} \\
&= ((0, -1), \arcsin(\cos(x))) && \text{if } \sin(x) < -1/\sqrt{2}
\end{aligned}$$

The *arc*, angle, function is defined by

$$\begin{aligned}
arc(x, y) &= -\arccos(x/\sqrt{x^2 + y^2}) && \text{if } y < 0 \\
&= \arccos(x/\sqrt{x^2 + y^2}) && \text{if } y \geq 0
\end{aligned}$$

5.3.8 Operations for radian trigonometric elementary functions

There shall be one radian big-angle parameter:

$$big_angle_r_F \in F$$

It should have the following default value:

$$big_angle_r_F = r_F^{\lceil p_F/2 \rceil}$$

A binding or implementation can include a method to change the value of the radian big-angle parameter. This method should only allow the value of this parameter to be set to a value greater than $2 \cdot \pi$ and such that $ulp_F(big_angle_r_F) < \pi/1000$.

NOTE – Part 1 requires only that $p_F \geq 2$, but see also A.5.2.0.2 in part 1. This part requires that $p_F \geq 2 \cdot \max\{1, \lceil \log_{r_F}(2 \cdot \pi) \rceil\}$, in order to allow at least the first two cycles (plus and minus) to be in the interval $[-big_angle_r_F, big_angle_r_F]$. In order to allow $ulp_F(big_angle_r_F) < \pi/1000$, $p_F \geq 2 + \lceil \log_{r_F}(1000) \rceil$ should hold.

For use in the approximation helper function's signatures, define

$$F^{2 \cdot \pi} = (F \cup \{n \cdot \pi/4, n \cdot \pi/6 \mid n \in \mathbb{Z}\}) \cap [-big_angle_r_F, big_angle_r_F]$$

There shall be three maximum error parameters for radian trigonometric operations:

$$max_error_rad_F \in F$$

$$max_error_sin_F \in F$$

$$max_error_tan_F \in F$$

The *max_error_rad_F* parameter shall have a value that is 0.5 (ulp). The *max_error_sin_F* parameter shall have a value that is $\leq 1.5 \cdot rnd_error_F$. The *max_error_tan_F* parameter shall have a value that is $\leq 2 \cdot rnd_error_F$. If the binding standard requires that the *max_error_rad_F* parameter has the value 0.5, that parameter need not be made available for programs.

5.3.8.1 Radian angle normalisation

The rad_F^* approximation helper function:

$$rad_F^* : F^{2\cdot\pi} \rightarrow \mathcal{R}$$

$rad_F^*(x)$ returns a close approximation to $rad(x)$ in $\mathcal{R}$, if $|x| \leq big_angle_r_F$, with maximum error $max_error_rad_F$.

The $axis_rad_F^*$ approximation helper function:

$$axis_rad_F^* : F^{2\cdot\pi} \rightarrow \{(1, 0), (0, 1), (-1, 0), (0, -1)\} \times \mathcal{R}$$

$axis_rad_F^*(x)$ returns a close approximation to $axis_rad(x)$, if $|x| \leq big_angle_r_F$, with maximum error $max_error_rad_F$ for the second part of the result. The approximation consists of that the second part of the result (the offset from the indicated axis) is approximate. The first part (the nearest axis indication) shall be exact if $|x| \leq big_angle_r_F$.

NOTE 1 – With the maximum error 0.5 ulp, these helper functions are not really needed. However, Annex A allows for partial conformity, such that the maximum error for these two helper functions may be greater than 0.5 ulp.

Further requirements on the rad_F^* and $axis_rad_F^*$ approximation helper functions are:

$$\begin{aligned} rad_F^*(x) &= x && \text{if } x \in F^{2\cdot\pi} \text{ and } |x| < \pi \\ snd(axis_rad_F^*(x)) &= rad_F^*(x) && \text{if } x \in F^{2\cdot\pi} \text{ and } fst(axis_rad_F^*(x)) = (1, 0) \end{aligned}$$

The rad_F operation:

$$\begin{aligned} rad_F : F &\rightarrow F \cup \{\text{underflow}, \text{absolute_precision_underflow}\} \\ rad_F(x) &= result_F^*(rad_F^*(x), nearest_F) && \text{if } x \in F \text{ and } |x| > fminN_F \text{ and } |x| \leq big_angle_r_F \\ &= x && \text{if } (x \in F \text{ and } |x| \leq fminN_F) \text{ or } x = -0 \\ &= \text{absolute_precision_underflow}(qNaN) && \text{if } x \in F \text{ and } |x| > big_angle_r_F \\ &= no_result_F(x) && \text{otherwise} \end{aligned}$$

The $axis_rad_F$ operation:

$$\begin{aligned} axis_rad_F : F &\rightarrow (\{(1, 0), (0, 1), (-1, 0), (0, -1)\} \times F) \cup \{\text{absolute_precision_underflow}\} \\ axis_rad_F(x) &= (fst(axis_rad_F^*(x)), result_F^*(snd(axis_rad_F^*(x)), nearest_F)) && \text{if } x \in F \text{ and } |x| > fminN_F \text{ and } |x| \leq big_angle_r_F \\ &= ((1, 0), x) && \text{if } (x \in F \text{ and } |x| \leq fminN_F) \text{ or } x = -0 \\ &= \text{absolute_precision_underflow}((qNaN, qNaN), qNaN) && \text{if } x \in F \text{ and } |x| > big_angle_r_F \\ &= ((qNaN, qNaN), qNaN) && \text{if } x \text{ is a quiet NaN} \\ &= \text{invalid}((qNaN, qNaN), qNaN) && \text{otherwise} \end{aligned}$$

NOTE 2 – rad_F is simpler, easier to use, but less accurate than $axis_rad_F$. The latter may still not be sufficient for implementing the radian trigonometric operations to less than the maximum error stated by the parameters.

5.3.8.2 Radian sine

The $\sin_F^*$ approximation helper function:

$$\sin_F^* : F^{2\cdot\pi} \rightarrow \mathcal{R}$$

$\sin_F^*(x)$ returns a close approximation to $\sin(x)$ in $\mathcal{R}$ if $|x| \leq \text{big_angle_r}_F$, with maximum error max_error_sin_F .

Further requirements on the $\sin_F^*$ approximation helper function are:

$$\begin{aligned} \sin_F^*(n \cdot 2 \cdot \pi + \pi/6) &= 1/2 & \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi/6| \leq \text{big_angle_r}_F \\ \sin_F^*(n \cdot 2 \cdot \pi + \pi/2) &= 1 & \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi/2| \leq \text{big_angle_r}_F \\ \sin_F^*(n \cdot 2 \cdot \pi + 5 \cdot \pi/6) &= 1/2 & \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + 5 \cdot \pi/6| \leq \text{big_angle_r}_F \\ \sin_F^*(x) &= x & \text{if } x \in F^{2\cdot\pi} \text{ and } \sin_F^*(x) \neq \sin(x) \text{ and} \\ & & |x| \leq \sqrt{3 \cdot \text{epsilon}_F / r_F} \\ \sin_F^*(-x) &= -\sin_F^*(x) & \text{if } x \in F^{2\cdot\pi} \end{aligned}$$

The $\sin_F$ operation:

$$\begin{aligned} \sin_F : F &\rightarrow F \cup \{\text{underflow}, \text{absolute_precision_underflow}\} \\ \sin_F(x) &= \text{result}_F^*(\sin_F^*(x), \text{nearest}_F) \\ &= \text{rad}_F(x) & \text{if } x \in F \text{ and } \text{fmin}_F < |x| \text{ and } |x| \leq \text{big_angle_r}_F \\ & & \text{otherwise} \end{aligned}$$

NOTE – **underflow** is here explicitly avoided for subnormal arguments, but the operation may **underflow** for other arguments.

5.3.8.3 Radian cosine

The $\cos_F^*$ approximation helper function:

$$\cos_F^* : F^{2\cdot\pi} \rightarrow \mathcal{R}$$

$\cos_F^*(x)$ returns a close approximation to $\cos(x)$ in $\mathcal{R}$ if $|x| \leq \text{big_angle_r}_F$, with maximum error max_error_sin_F .

Further requirements on the $\cos_F^*$ approximation helper function are:

$$\begin{aligned} \cos_F^*(n \cdot 2 \cdot \pi) &= 1 & \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi| \leq \text{big_angle_r}_F \\ \cos_F^*(n \cdot 2 \cdot \pi + \pi/3) &= 1/2 & \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi/3| \leq \text{big_angle_r}_F \\ \cos_F^*(n \cdot 2 \cdot \pi + 2 \cdot \pi/3) &= -1/2 & \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + 2 \cdot \pi/3| \leq \text{big_angle_r}_F \\ \cos_F^*(n \cdot 2 \cdot \pi + \pi) &= -1 & \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi| \leq \text{big_angle_r}_F \\ \cos_F^*(x) &= 1 & \text{if } x \in F^{2\cdot\pi} \text{ and } \cos_F^*(x) \neq \cos(x) \text{ and} \\ & & |x| < \sqrt{\text{epsilon}_F / r_F} \\ \cos_F^*(-x) &= \cos_F^*(x) & \text{if } x \in F^{2\cdot\pi} \end{aligned}$$

The $\cos_F$ operation:

$$\begin{aligned} \cos_F : F &\rightarrow F \cup \{\text{underflow}, \text{absolute_precision_underflow}\} \\ \cos_F(x) &= \text{result}_F^*(\cos_F^*(x), \text{nearest}_F) \\ &= 1 & \text{if } x \in F \text{ and } |x| \leq \text{big_angle_r}_F \\ &= \text{rad}_F(x) & \text{if } x = -0 \\ & & \text{otherwise} \end{aligned}$$

5.3.8.4 Radian tangent

The $\tan_F^*$ approximation helper function:

$$\tan_F^* : F^{2 \cdot \pi} \rightarrow \mathcal{R}$$

$\tan_F^*(x)$ returns a close approximation to $\tan(x)$ in $\mathcal{R}$ if $|x| \leq \text{big_angle_}r_F$, with maximum error $\text{max_error_}\tan_F$.

Further requirements on the $\tan_F^*$ approximation helper function are:

$$\begin{aligned} \tan_F^*(n \cdot 2 \cdot \pi + \pi/4) &= 1 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi/4| \leq \text{big_angle_}r_F \\ \tan_F^*(n \cdot 2 \cdot \pi + 3 \cdot \pi/4) &= -1 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + 3 \cdot \pi/4| \leq \text{big_angle_}r_F \\ \tan_F^*(x) &= x && \text{if } x \in F^{2 \cdot \pi} \text{ and } \tan_F^*(x) \neq \tan(x) \text{ and} \\ &&& |x| < \sqrt{\text{epsilon}_F/r_F} \\ \tan_F^*(-x) &= -\tan_F^*(x) && \text{if } x \in F^{2 \cdot \pi} \end{aligned}$$

NOTE 1 – $\tan$ has a smallest period of π , but the above expresses a period of $2 \cdot \pi$, which is more in line with the other operations. The desired points of extra accuracy are still covered.

The $\tan_F$ operation:

$$\begin{aligned} \tan_F : F &\rightarrow F \cup \{\text{underflow}, \text{overflow}, \text{absolute_precision_underflow}\} \\ \tan_F(x) &= \text{result}_F^*(\tan_F^*(x), \text{nearest}_F) && \text{if } x \in F \text{ and } \text{fmin}_F < |x| \text{ and } |x| \leq \text{big_angle_}r_F \\ &= \text{rad}_F(x) && \text{otherwise} \end{aligned}$$

NOTE 2 – **underflow** is explicitly avoided for subnormal arguments, but the operation may **underflow** for other arguments.

5.3.8.5 Radian cotangent

The $\cot_F^*$ approximation helper function:

$$\cot_F^* : F^{2 \cdot \pi} \rightarrow \mathcal{R}$$

$\cot_F^*(x)$ returns a close approximation to $\cot(x)$ in $\mathcal{R}$ if $|x| \leq \text{big_angle_}r_F$, with maximum error $\text{max_error_}\tan_F$.

Further requirements on the $\cot_F^*$ approximation helper function are:

$$\begin{aligned} \cot_F^*(n \cdot 2 \cdot \pi + \pi/4) &= 1 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi/4| \leq \text{big_angle_}r_F \\ \cot_F^*(n \cdot 2 \cdot \pi + 3 \cdot \pi/4) &= -1 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + 3 \cdot \pi/4| \leq \text{big_angle_}r_F \\ \cot_F^*(-x) &= -\cot_F^*(x) && \text{if } x \in F^{2 \cdot \pi} \end{aligned}$$

NOTE – $\cot$ has a smallest period of π , but the above expresses a period of $2 \cdot \pi$, which is more in line with the other operations. The desired points of extra accuracy are still covered.

The $\cot_F$ operation:

$$\begin{aligned} \cot_F : F &\rightarrow F \cup \{\text{underflow}, \text{overflow}, \text{infinitary}, \text{absolute_precision_underflow}\} \\ \cot_F(x) &= \text{result}_F^*(\cot_F^*(x), \text{nearest}_F) && \text{if } x \in F \text{ and } x \neq 0 \text{ and } |x| \leq \text{big_angle_}r_F \\ &= \text{infinitary}(+\infty) && \text{if } x = 0 \\ &= \text{infinitary}(-\infty) && \text{if } x = -0 \\ &= \text{rad}_F(x) && \text{otherwise} \end{aligned}$$

5.3.8.6 Radian secant

The $\sec_F^*$ approximation helper function:

$$\sec_F^* : F^{2\cdot\pi} \rightarrow \mathcal{R}$$

$\sec_F^*(x)$ returns a close approximation to $\sec(x)$ in $\mathcal{R}$ if $|x| \leq \text{big_angle_r}_F$, with maximum error max_error_tan_F .

Further requirements on the $\sec_F^*$ approximation helper function are:

$$\begin{aligned} \sec_F^*(n \cdot 2 \cdot \pi) &= 1 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi| \leq \text{big_angle_r}_F \\ \sec_F^*(n \cdot 2 \cdot \pi + \pi/3) &= 2 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi/3| \leq \text{big_angle_r}_F \\ \sec_F^*(n \cdot 2 \cdot \pi + 2 \cdot \pi/3) &= -2 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + 2 \cdot \pi/3| \leq \text{big_angle_r}_F \\ \sec_F^*(n \cdot 2 \cdot \pi + \pi) &= -1 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi| \leq \text{big_angle_r}_F \\ \sec_F^*(x) &= 1 && \text{if } x \in F^{2\cdot\pi} \text{ and } \sec_F^*(x) \neq \sec(x) \text{ and } \\ &&& |x| < \sqrt{\text{epsilon}_F} \\ \sec_F^*(-x) &= \sec_F^*(x) && \text{if } x \in F^{2\cdot\pi} \end{aligned}$$

The $\sec_F$ operation:

$$\begin{aligned} \sec_F : F &\rightarrow F \cup \{\text{overflow}, \text{absolute_precision_underflow}\} \\ \sec_F(x) &= \text{result}_F^*(\sec_F^*(x), \text{nearest}_F) && \text{if } x \in F \text{ and } |x| \leq \text{big_angle_r}_F \\ &= 1 && \text{if } x = -\mathbf{0} \\ &= \text{rad}_F(x) && \text{otherwise} \end{aligned}$$

5.3.8.7 Radian cosecant

The $\csc_F^*$ approximation helper function:

$$\csc_F^* : F^{2\cdot\pi} \rightarrow \mathcal{R}$$

$\csc_F^*(x)$ returns a close approximation to $\csc(x)$ in $\mathcal{R}$ if $|x| \leq \text{big_angle_r}_F$, with maximum error max_error_tan_F .

Further requirements on the $\csc_F^*$ approximation helper function are:

$$\begin{aligned} \csc_F^*(n \cdot 2 \cdot \pi + \pi/6) &= 2 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi/6| \leq \text{big_angle_r}_F \\ \csc_F^*(n \cdot 2 \cdot \pi + \pi/2) &= 1 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + \pi/2| \leq \text{big_angle_r}_F \\ \csc_F^*(n \cdot 2 \cdot \pi + 5 \cdot \pi/6) &= 2 && \text{if } n \in \mathcal{Z} \text{ and } |n \cdot 2 \cdot \pi + 5 \cdot \pi/6| \leq \text{big_angle_r}_F \\ \csc_F^*(-x) &= -\csc_F^*(x) && \text{if } x \in F^{2\cdot\pi} \end{aligned}$$

The $\csc_F$ operation:

$$\begin{aligned} \csc_F : F &\rightarrow F \cup \{\text{overflow}, \text{infinitary}, \text{absolute_precision_underflow}\} \\ \csc_F(x) &= \text{result}_F^*(\csc_F^*(x), \text{nearest}_F) && \text{if } x \in F \text{ and } x \neq 0 \text{ and } |x| \leq \text{big_angle_r}_F \\ &= \text{infinitary}(+\infty) && \text{if } x = 0 \\ &= \text{infinitary}(-\infty) && \text{if } x = -\mathbf{0} \\ &= \text{rad}_F(x) && \text{otherwise} \end{aligned}$$

5.3.8.8 Radian cosine with sine

$$\begin{aligned} \text{cossin}_F : F &\rightarrow (F \times F) \cup \{\text{underflow}, \text{absolute_precision_underflow}\} \\ \text{cossin}_F(x) &= (\cos_F(x), \sin_F(x)) \end{aligned}$$

NOTES

- 1 If there is an **absolute_precision_underflow** notification, then both result parts suffer from the **absolute_precision_underflow** and the continuation values for both parts are **qNaN**. Similarly for NaN and infinitary arguments.
- 2 If there is an **underflow** notification, only one of the result parts suffer from the underflow, and the other part has an absolute value greater than $fminN_F$.

5.3.8.9 Radian arc sine

The arcsin_F^* approximation helper function:

$$\text{arcsin}_F^* : F \rightarrow \mathcal{R}$$

$\text{arcsin}_F^*(x)$ returns a close approximation to $\arcsin(x)$ in $\mathcal{R}$, with maximum error max_error_sin_F .

Further requirements on the arcsin_F^* approximation helper function are:

$$\begin{aligned} \text{arcsin}_F^*(1/2) &= \pi/6 \\ \text{arcsin}_F^*(1) &= \pi/2 \\ \text{arcsin}_F^*(x) &= x && \text{if } x \in F \text{ and } \text{arcsin}_F^*(x) \neq \arcsin(x) \text{ and } |x| < \sqrt{2 \cdot \text{epsilon}_F / r_F} \\ \text{arcsin}_F^*(-x) &= -\text{arcsin}_F^*(x) && \text{if } x \in F \end{aligned}$$

The $\text{arcsin}_F^\#$ range limitation helper function (for $x \in F$):

$$\text{arcsin}_F^\#(x) = \max\{up_F(-\pi/2), \min\{\text{arcsin}_F^*(x), down_F(\pi/2)\}\}$$

The arcsin_F operation:

$$\begin{aligned} \text{arcsin}_F : F &\rightarrow F \cup \{\text{invalid}\} \\ \text{arcsin}_F(x) &= \text{result}_F^*(\text{arcsin}_F^\#(x), \text{nearest}_F) && \text{if } x \in F \text{ and } fminN_F < |x| \leq 1 \\ &= x && \text{if } (x \in F \text{ and } |x| \leq fminN_F) \text{ or } x = -0 \\ &= no_result_F(x) && \text{otherwise} \end{aligned}$$

NOTE – **underflow** is explicitly avoided.

5.3.8.10 Radian arc cosine

The arccos_F^* approximation helper function:

$$\text{arccos}_F^* : F \rightarrow \mathcal{R}$$

$\text{arccos}_F^*(x)$ returns a close approximation to $\arccos(x)$ in $\mathcal{R}$, with maximum error max_error_sin_F .

Further requirements on the arccos_F^* approximation helper function are:

$$\begin{aligned}
\arccos_F^*(1/2) &= \pi/3 \\
\arccos_F^*(0) &= \pi/2 \\
\arccos_F^*(-1/2) &= 2 \cdot \pi/3 \\
\arccos_F^*(-1) &= \pi
\end{aligned}$$

The $\arccos_F^\#$ range limitation helper function (for $x \in F$):

$$\arccos_F^\#(x) = \min\{\arccos_F^*(x), \text{down}_F(\pi)\}$$

The $\arccos_F$ operation:

$$\begin{aligned}
\arccos_F : F &\rightarrow F \cup \{\text{invalid}\} \\
\arccos_F(x) &= \text{result}_F^*(\arccos_F^\#(x), \text{nearest}_F) \\
&\quad \text{if } x \in F \text{ and } |x| \leq 1 \\
&= \arccos_F(0) \quad \text{if } x = -\mathbf{0} \\
&= \text{no_result}_F(x) \quad \text{otherwise}
\end{aligned}$$

5.3.8.11 Radian arc tangent

The $\arctan_F^*$ approximation helper function:

$$\arctan_F^* : F \rightarrow \mathcal{R}$$

$\arctan_F^*(x)$ returns a close approximation to $\arctan(x)$ in $\mathcal{R}$, with maximum error max_error_tan_F .

Further requirements on the $\arctan_F^*$ approximation helper function are:

$$\begin{aligned}
\arctan_F^*(1) &= \pi/4 \\
\arctan_F^*(x) &= x && \text{if } x \in F \text{ and } \arctan_F^*(x) \neq \arctan(x) \text{ and } |x| \leq \sqrt{1.5 \cdot \text{epsilon}_F / r_F} \\
\arctan_F^*(x) &= \pi/2 && \text{if } x \in F \text{ and } \arctan_F^*(x) \neq \arctan(x) \text{ and } x > 3 \cdot r_F / \text{epsilon}_F \\
\arctan_F^*(-x) &= -\arctan_F^*(x) && \text{if } x \in F
\end{aligned}$$

The $\arctan_F^\#$ range limitation helper function (for $x \in F$):

$$\arctan_F^\#(x) = \max\{\text{up}_F(-\pi/2), \min\{\arctan_F^*(x), \text{down}_F(\pi/2)\}\}$$

The $\arctan_F$ operation:

$$\begin{aligned}
\arctan_F : F &\rightarrow F \\
\arctan_F(x) &= \text{result}_F^*(\arctan_F^\#(x), \text{nearest}_F) \\
&\quad \text{if } x \in F \text{ and } \text{fmin}N_F < |x| \\
&= x && \text{if } (x \in F \text{ and } |x| \leq \text{fmin}N_F) \text{ or } x = -\mathbf{0} \\
&= \text{up}_F(-\pi/2) && \text{if } x = -\infty \\
&= \text{down}_F(\pi/2) && \text{if } x = +\infty \\
&= \text{no_result}_F(x) && \text{otherwise}
\end{aligned}$$

NOTES

- 1 $\arctan_F(x) \approx \text{arc}_F(1, x)$. (arc_F is specified in subclause 5.3.8.15 below.)
- 2 **underflow** is explicitly avoided.

5.3.8.12 Radian arc cotangent

This clause specifies two inverse cotangent operations. One approximating the sign symmetric (but discontinuous at 0) arccot , the other approximating the continuous (but not sign symmetric) $\operatorname{arccotc}$.

The $\operatorname{arccot}_F^*$ approximation helper function:

$$\operatorname{arccot}_F^* : F \rightarrow \mathcal{R}$$

$\operatorname{arccot}_F^*(x)$ returns a close approximation to $\operatorname{arccot}(x)$ in $\mathcal{R}$, with maximum error $\max_error_tan_F$.

The $\operatorname{arccotc}_F^*$ approximation helper function:

$$\operatorname{arccotc}_F^* : F \rightarrow \mathcal{R}$$

$\operatorname{arccotc}_F^*(x)$ returns a close approximation to $\operatorname{arccotc}(x)$ in $\mathcal{R}$, with maximum error $\max_error_tan_F$.

Further requirements on the $\operatorname{arccot}_F^*$ and $\operatorname{arccotc}_F^*$ approximation helper functions are:

$$\begin{aligned} \operatorname{arccot}_F^*(1) &= \pi/4 \\ \operatorname{arccot}_F^*(0) &= \pi/2 \\ \operatorname{arccot}_F^*(-x) &= -\operatorname{arccot}_F^*(x) && \text{if } x \in F \text{ and } x \neq 0 \\ \\ \operatorname{arccotc}_F^*(x) &= \operatorname{arccot}_F^*(x) && \text{if } x \in F \text{ and } x \geq 0 \\ \operatorname{arccotc}_F^*(-1) &= 3 \cdot \pi/4 \\ \operatorname{arccotc}_F^*(x) &= \pi && \text{if } x \in F \text{ and } \operatorname{arccotc}_F^*(x) \neq \operatorname{arccotc}(x) \text{ and } \\ &&& x < -3 \cdot r_F / \epsilon_F \end{aligned}$$

The $\operatorname{arccot}_F^\#$ and $\operatorname{arccotc}_F^\#$ range limitation helper functions (for $x \in F$):

$$\begin{aligned} \operatorname{arccot}_F^\#(x) &= \max\{up_F(-\pi/2), \min\{\operatorname{arccot}_F^*(x), down_F(\pi/2)\}\} \\ \operatorname{arccotc}_F^\#(x) &= \min\{\operatorname{arccotc}_F^*(x), down_F(\pi)\} \end{aligned}$$

The arccot_F operation:

$$\begin{aligned} \operatorname{arccot}_F : F &\rightarrow F \cup \{\mathbf{underflow}\} \\ \operatorname{arccot}_F(x) &= \operatorname{result}_F^*(\operatorname{arccot}_F^\#(x), \operatorname{nearest}_F) && \text{if } x \in F \\ &= up_F(-\pi/2) && \text{if } x = -\mathbf{0} \\ &= -\mathbf{0} && \text{if } x = -\infty \\ &= 0 && \text{if } x = +\infty \\ &= \operatorname{no_result}_F(x) && \text{otherwise} \end{aligned}$$

NOTES

- 1 $\operatorname{arccot}_F(\operatorname{neg}_F(x)) = \operatorname{neg}_F(\operatorname{arccot}_F(x))$.
- 2 Due to the range limitation, $\operatorname{arccot}_F(0)$ need not equal $\operatorname{arccotc}_F(0)$.

The $\operatorname{arccotc}_F$ operation:

$$\begin{aligned} \operatorname{arccotc}_F : F &\rightarrow F \cup \{\mathbf{underflow}\} \\ \operatorname{arccotc}_F(x) &= \operatorname{result}_F^*(\operatorname{arccotc}_F^\#(x)) && \text{if } x \in F \\ &= \operatorname{nearest}_F(\pi/2) && \text{if } x = -\mathbf{0} \\ &= down_F(\pi) && \text{if } x = -\infty \\ &= 0 && \text{if } x = +\infty \\ &= \operatorname{no_result}_F(x) && \text{otherwise} \end{aligned}$$