SC22/WG11 N443

INTERNATIONAL STANDARD

ISO/IEC 10967-2

Third Committee Draft
1998-09-18

Information technology — Language independent arithmetic —

Part 2: Elementary numerical functions

Technologies de l'information —
Arithmétique indépendante de langage —

Partie 2: Fonctions numériques élémentaires

THIRD COMMITTEE DRAFT
September 18, 1998 10:29

Editor:
Kent Karlsson
IMI, Industri-Matematik International
Kungsgatan 12
SE-411 19 Göteborg
SWEDEN
Telephone: +46-31 10 22 44
Facsimile: +46-31 13 13 25
E-mail: keka@im.se

Reference number: ISO/IEC CD 10967-2.3:1998(E)

Contents

Foreword

ISO (the International Organisation for Standardisation) and IEC (the International Electrotechnical Commission) form the specialised system for world-wide standardisation. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organisation to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organisations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. 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.

International Standard ISO/IEC 10967-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Sub-Committee SC 22, *Programming languages*.

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 floating point arithmetic and complex elementary numerical functions*

Additional parts will specify other arithmetic datatypes or arithmetic operations.

Notes and annexes A to D of ISO/IEC 10967-2 are for information only.

Introduction

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.

Part 1 of ISO/IEC 10967, LIA-1, specifies the basic properties of integer and floating point types that can be relied upon in writing portable software.

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

The content of LIA-2 is based on LIA-1, and extends LIA-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. LIA-2 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. LIA-2 also includes specifications for a number of other functions deemed useful, even though they may not be stipulated by language standards.

The numerical functions covered by LIA-2 are computer approximations to mathematical functions of one or more real arguments. Accuracy versus performance requirements often vary with the application at hand. LIA-2 recognises this 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.

Annex A is intended to be read in parallel with the standard.

Information technology – Language independent arithmetic – Part 2: Elementary numerical functions

1 Scope

ISO/IEC 10967-2 defines the properties of numerical approximations for many of the real elementary numerical functions available in standard libraries for a variety of 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 ISO/IEC 10967-2. It is the computing environment, as seen by the programmer/user, that does or does not conform to the specifications.

The term *implementation* of ISO/IEC 10967-2 denotes the total computing environment, including hardware, language processors, subroutine libraries, exception handling facilities, other software, and all pertinent documentation.

1.1 Specifications included in ISO/IEC 10967-2

The specifications of ISO/IEC 10967-1 are included by reference in ISO/IEC 10967-2.

ISO/IEC 10967-2 provides specifications for numerical functions for which all operand values are of integer or floating point datatypes satisfying the requirements of ISO/IEC 10967-1. Boundaries for the occurrence of exceptions and the maximum error allowed are prescribed for each such operation. Also the result produced by a special value operand, such as an infinity, a **NaN**, or a (returnable) value in $\mathcal{R}$ is prescribed for each operation.

ISO/IEC 10967-2 covers most numerical functions required by the ISO standards for Ada, Basic, C/C++, Fortran, Extended Pascal, ISLisp, and PL/I. 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, hyperbolics, and

d) trigonometrics.

ISO/IEC 10967-2 also provide specifications for

e) conversions between implemented datatypes (possibly based on different radices) conforming to the requirements of ISO/IEC 10967-1,

f) the radix conversion operations used, for example, in text input and output.

In addition, it provides specifications for

g) the results produced when one or more operand value is an IEC 559 special value, and

h) program-visible parameters that characterise the operations.

ISO/IEC 10967-2 uses the same procedures as ISO/IEC 10967-1 for reporting errors.

1.2 Specifications not within the scope of ISO/IEC 10967-2

This standard provides no specifications for:

a) Numerical functions whose operands are of more than one datatype (with one exception). This standard neither requires nor excludes the presence of such "mixed operand" operations.

b) An interval data type, or the operations on such data. This standard neither requires nor excludes such data or operations.

c) A fixed point data type, or the operations on such data. This standard neither requires nor excludes such data or operations.

d) A rational data type, or the operations on such data. This standard neither requires nor excludes such data or operations.

e) The properties of arithmetic data types that are not related to the numerical process, such as the representation of values on physical media.

f) The properties of integer and floating point data types that properly belong in language standards. Examples include

   1) the syntax of literals and expressions,

   2) the precedence of operators,

   3) the rules of assignment and parameter passing,

   4) the presence or absence of automatic type coercions,

   5) the consequences of applying an operation to values of improper type, or to uninitialised data.

Nor does this part of ISO/IEC 10967 provide any specifications for

g) how numerical functions should be implemented,

h) which algorithms are to be used for the various operations,

i) the textual form used for input or output by any specific programming language,

j) complex, matrix, statistical, or symbolic 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 language standards and in language binding standards.

A binding standard specifies the correspondence between one or more operations and parameters defined in ISO/IEC 10967-2 and the concrete language syntax of some programming language. More generally, a binding standard specifies the correspondence between certain operations and the elements of some arbitrary computing entity. A language standard that explicitly provides such binding information can serve as a binding standard.

Conformity to ISO/IEC 10967-2 is always with respect to a specified set of operations. Conformity to ISO/IEC 10967-2 implies conformity to ISO/IEC 10967-1 for the integer and floating point datatypes used.

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

When a binding standard covers only a subset of the operations defined in ISO/IEC 10967-2, an implementation remains free to conform to ISO/IEC 10967-2 with respect to other operations independently of that binding standard.

When no binding standard for a language and some operations specified in ISO/IEC 10967-2 exists, an implementation conforms to this part of ISO/IEC 10967 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.

An implementation is free to provide operations that do not conform to ISO/IEC 10967-2 or that are beyond the scope of this Part. The implementation shall not claim or imply conformity with respect to such operations.

An implementation is permitted to have modes of operation that do not conform to ISO/IEC 10967-2. A conforming implementation shall specify how to select the modes of operation that ensure conformity.

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 ISO/IEC 10967-2 will include (explicitly or by reference) a binding for ISO/IEC 10967-1 as well, which in turn includes (explicitly or by reference) a binding for IEC 559 as well.

3 It is not possible to conform to ISO/IEC 10967-2 without specifying to which set of operations conformity is claimed.

3 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of ISO/IEC 10967-2. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on ISO/IEC 10967-2 are encouraged to investigate the possibility of applying the most recent edition of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

IEC 559:1989, *Binary floating-point arithmetic for microprocessor systems.*

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

4 Symbols and definitions

4.1 Symbols

In ISO/IEC 10967-2, $\mathcal{Z}$ denotes the set of mathematical integers, $\mathcal{R}$ denotes the set of classical real numbers, and $\mathcal{C}$ denotes the set of complex numbers. Note that $\mathcal{Z} \subset \mathcal{R} \subset \mathcal{C}$.

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

All prefix and infix operators have their conventional (exact) mathematical meaning. The conventional notation for set definition and manipulation is also used. In particular ISO/IEC 10967-2 uses

$\Rightarrow$ and $\Longleftrightarrow$ for logical implication and equivalence
$+$, $-$, $*$, $/$, x^y, $\log_x(y)$, $\sqrt{x}$, $|x|$, $\lfloor x \rfloor$, $\lceil x \rceil$, and round(x) on reals
$<$, $\leq$, $=$, $\neq$, $\geq$, and $>$ between reals
$\cup$, $\cap$, $\times$, $\in$, $\subset$, and $=$ on sets
max and min on non-empty sets of integers and reals
$\rightarrow$ for a mapping between sets

ISO/IEC 10967-2 uses $*$ for multiplication, and $\times$ for the Cartesian product of sets. $\sqrt{x} \in [0, \infty[$, when the function is defined. 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 round(x) designates the integer closest to x:

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

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

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

$$i|j \iff (i \neq 0 \text{ and } i * n = j \text{ for some } n \in \mathcal{Z})$$

The following ideal mathematical functions are defined in Chapter 4 of the *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* [27].

e^x, x^y,
ln, $\log_b$,
sinh, cosh, tanh, coth, sech, csch,
arcsinh, arccosh, arctanh, arccoth, arcsech, arccsch,
sin, cos, tan, cot, sec, csc,
arcsin, arccos, arctan, arccot, arcsec, arccsc.

Many of the inverses are multi-valued. The selection of which value to return, so as to make the inverses into functions, is done in the conventional way. The only one over which there is some difference of conventions it the arccot function. Conventions there vary for negative arguments; either a positive return value (giving a function that is continuous over zero), or a negative value (giving a sign symmetric function). In this part of ISO/IEC 10967, arccot refers to the continuous inverse function, and arcctg refers to the sign symmetric inverse function.

NOTE 1 – *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* [27] uses the notation arccot for what is called arcctg in LIA-2.

Define the following mathematical functions:

$rad : \mathcal{R} \rightarrow \mathcal{R}$
$axis\_rad : \mathcal{R} \rightarrow \{(1,0), (0,1), (-1,0), (0,-1)\} \times \mathcal{R}$
$arc : \mathcal{R} \times \mathcal{R} \rightarrow \mathcal{R}$

The *rad* function is defined by

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

The *axis_rad* function is defined by

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

The *arc* function is defined by

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

NOTES

2 $rad(x) = \arccos(\cos(x))$ if $\sin(x) > 0$ and $rad(x) = -\arccos(\cos(x))$ if $\sin(x) < 0$.

3 The first part of $axis\_rad(x)$ indicates which axis is nearest to the angle x. The second part of $axis\_rad(x)$ is an angle offset from the axis that is nearest to the angle x. The second part of $axis\_rad(x)$ is equal to $rad(x)$ if $\cos(x) \geq 1/\sqrt{2}$ (i.e. if the first part of $axis\_rad(x)$ is $(1,0)$). More generally, the second part of $axis\_rad(x)$ is equal to $rad(4*x)/4$.

4 $rad(x)$ returns the same angle as the angle value x, but the returned angle value is between $-\pi$ and π. The *rad* function is defined to be used as the basis for the angle normalisation operations. The *axis_rad* function is defined to be used as the basis for a numerically more accurate radian angle normalisation operation. The *arc* function is defined to be used as the basis for the arcus operations, which are used for conversion from Cartesian to polar co-ordinates.

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

$fst((x,y)) = x$, and $snd((x,y)) = y$.

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: $x : [x_1, ..., x_n] = [x, x_1, ..., x_n]$

$[S]$, where S is a set, denotes the set of finite sequences, where each value in each sequence is in S.

NOTE 5 – It should be clear from context if $[X]$ is a sequence of one element, or the set of sequences with values from X. It should also be 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 ISO/IEC 10967-1.

The following symbols are defined in ISO/IEC 10967-1:1994, and used in this part.

Exceptional values:
integer_overflow, **floating_overflow**, **underflow**, and **undefined**.

Integer parameters:
$bounded_I$, $maxint_I$, and $minint_I$.

Integer helper functions:
$wrap_I$.

Integer operations:
neg_I, add_I, sub_I, mul_I, rem_I^f.

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 constants:

rnd_style_F, rnd_error_F.

Floating point value sets related to F:

F^*, F_D, F_N.

Floating point helper functions:

e_F, $result_F$.

Floating point operations:

neg_F, add_F, sub_F, mul_F, div_F, $sign_F$.

Floating point conversion operations:

$cvt_{F \to F'}$.

Three new exceptional values, **invalid**, **pole**, and **angle_too_big**, are introduced in ISO/IEC 10967-2 in addition to those in ISO/IEC 10967-1:1994. **invalid** and **pole** are in ISO/IEC 10967-2 used instead of the **undefined** of ISO/IEC 10967-1:1994. **angle_too_big** is used when the floating point angle value argument is so big that even an highly accurate result from a trigonometric operation is questionable, due to that the density of floating point values has decreased significantly at these big angle values.

NOTE 6 – ISO/IEC 10967-2 provides specifications for angle normalisation operations that can be used to transform a (not too big) angle value to an angle value within one cycle for the same (or very close) angle.

The following symbols represent values defined in IEC 559:1989 and used in ISO/IEC 10967-2:

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

These symbols are not part of 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 7 – ISO/IEC 10967-2 uses the above four symbols for compatibility with IEC 559. In particular, the symbol **−0** is not the application of (mathematical) unary − to the value 0, and is a value logically distinct from 0.

4.2 Definitions

For the purposes of ISO/IEC 10967-2, the following definitions apply:

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

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

NOTE 1 – This standard specifies requirements for integer and floating point data types. Complex numbers are not covered here, but will be included in a subsequent part of ISO/IEC 10967 [15].

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. (Contrast with *exceptional value*. See 6.1.2 of ISO/IEC 10967-1:1994.)

datatype: A set of values and a set of operations that manipulate those values.

denormalisation loss: A larger than normal rounding error caused by the fact that subnormal values have less than full precision. (See 5.2.5 of ISO/IEC 10967-1:1994, for a full definition.)

denormalised, denormal: 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 5.2 of ISO/IEC 10967-1:1994, 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 numeric result. This might arise because no such result exists mathematically, or because the mathematical result cannot be represented with sufficient accuracy.

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 ISO/IEC 10967-1:1994.)

NOTES

2 Exceptional values are used as part of the defining formalism only. With respect to ISO/IEC 10967, 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.

3 Exceptional values are not to be confused with the NaNs and infinities defined in IEC 559. 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 of ISO/IEC 10967)**:** The total arithmetic environment presented to a programmer, including hardware, language processors, exception handling facilities, subroutine libraries, other software, and all pertinent documentation.

literal: A syntactic entity denoting a value without having proper sub-entities that are expressions.

monotonic approximation: An operation $op_F : ... \times F \times ... \to F$, where the other arguments are kept constant, is a monotonic approximation of a predetermined mathematical function $h : \mathcal{R} \to \mathcal{R}$ if, for every $a \in F$ and $b \in F$,

a) h is monotonic non-decreasing on $[a, b]$ implies $op_F(..., a, ...) \leq op_F(..., b, ...)$,

b) h is monotonic non-increasing on $[a, b]$ implies $op_F(..., a, ...) \geq op_F(..., b, ...)$.

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

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

normalised: The non-zero values of a floating point type F that provide the full precision allowed by that type. (See F_N in 5.2 of ISO/IEC 10967-1:1994 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 ISO/IEC 10967-1:1994 for details.)

numeral: A numeric literal. It may denote a value in $\mathcal{R}$, 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 user/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, monotonous, and continuous in at least one side of the neighbourhood of x_0, and $\lim_{x \to x_0} f(x)$ is infinite.

precision: The number of digits in the fraction of a floating point number. (See 5.2 of ISO/IEC 109671:1994.)

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 5.2.6 of ISO/IEC 10967-1:1994). (See also A.5.2.6 of ISO/IEC 10967-1:1994 for some examples.)

rounding function: Any function $rnd : \mathcal{R} \to X$ (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 $\mathcal{R}$,

$$x \in X \Rightarrow rnd(x) = x$$
$$x < y \Rightarrow rnd(x) \leq rnd(y)$$

Note that if $u \in \mathcal{R}$ 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.

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 [2].)

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 [2].)

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.

The signature

$$add_I : I \times I \to I \cup \{\textbf{integer\_overflow}\}$$

states that the operation named add_I shall accept any pair of I values as input, and (when given such input) shall return either a single I value as its output or the exceptional value **integer_overflow**.

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 input. An operation given an argument outside the stipulated argument domain may produce a result outside the stipulated results range.

The signature chosen in the specifications below is the one that allows all non-special values as input, and gives all non-special, special, and exceptional values that may result. More restrictive (for example, only the domain for which non-exceptional values result) or less restrictive (for example, including IEC 559 special values as arguments) are not given in the specifications below.

subnormal: A denormal value, the value 0, or the value **−0**.

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, and radix r, is r^{t-p}, and the ulp of a subnormal value is $fminD_F$. (See 5.2 of ISO/IEC 10967-1:1994.)

5 Specifications for the numerical functions

5.1 Additional basic integer operations

Clause 5.1 of ISO/IEC 10967-1 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 an integer datatype conforming to ISO/IEC 10967-1. Integer datatypes conforming to ISO/IEC 10967-1 usually 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. String formats for integer values usually do contain (signalling) **NaN**s, however, when that string format is regarded as an (non-ISO/IEC 10967-1) integer datatype. See clause 5.4 on conversions.

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

The $result_I$ helper function:

$$result_I : \mathcal{Z} \to I \cup \{\textbf{integer\_overflow}\}$$

$$\begin{array}{lll} result_I(x) & = x & \text{if } x \in I \\ & = \textbf{integer\_overflow} & \text{if } x \notin I \text{ and } x \in \mathcal{Z} \end{array}$$

The $wrap_I$ helper function (also used in ISO/IEC 10967-1). $maxint_I$ and $minint_I$ are from ISO/IEC 10967-1.

$$wrap_I : \mathcal{Z} \to I$$

$$\begin{array}{lll} wrap_I(x) & = x - (n * (maxint_I - minint_I + 1)) & \\ & & \text{if } x \in \mathcal{Z} \text{ and } I \neq \mathcal{Z} \\ & = x & \text{otherwise} \end{array}$$

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 $I \neq \mathcal{Z}$;
$n = \lceil (x - maxint_I)/(maxint_I - minint_I + 1) \rceil$ if $x \in \mathcal{Z}$ and $I \neq \mathcal{Z}$.

2 For some wrapping basic arithmetic operations this n is computed by the '*_ov*' operations in clause 5.1.9.

5.1.2 Integer maximum and minimum operations

$max_I : I \times I \to I$

$max_I(x, y) \quad = \max\{x, y\} \qquad \text{if } x, y \in I$

$min_I : I \times I \to I$

$min_I(x, y) \quad = \min\{x, y\} \qquad \text{if } x, y \in I$

$max\_seq_I : [I] \to I \cup \{\textbf{invalid}\}$

$max\_seq_I([x_1, ..., x_n])$

$= \textbf{invalid} \qquad \text{if } n = 0 \text{ and } -\infty \text{ is not available}$

$= \max\{x_1, ..., x_n\} \qquad \text{if } n \geq 1 \text{ and } \{x_1, ..., x_n\} \subset I$

$min\_seq_I : [I] \to I \cup \{\textbf{invalid}\}$

$min\_seq_I([x_1, ..., x_n])$

$= \textbf{invalid} \qquad \text{if } n = 0 \text{ and } +\infty \text{ is not available}$

$= \min\{x_1, ..., x_n\} \qquad \text{if } n \geq 1 \text{ and } \{x_1, ..., x_n\} \subset I$

5.1.3 Integer positive difference (monus, diminish) operation

$dim_I : I \times I \to I \cup \{\textbf{integer\_overflow}\}$

$dim_I(x, y) \quad = result_I(\max\{0, x - y\}) \quad \text{if } x, y \in I$

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

5.1.4 Integer power and arithmetic shift operations

$power_I : I \times I \to I \cup \{\textbf{integer\_overflow}, \textbf{invalid}\}$

$power_I(x, y) \quad = result_I(x^y) \qquad \text{if } x, y \in I \text{ and } y > 0$

$= 1 \qquad \text{if } x \in I \text{ and } y = 0 \text{ and } x \neq 0$

$= \textbf{invalid}(1) \qquad \text{if } y = 0 \text{ and } x = 0$

$= \textbf{invalid} \qquad \text{if } x, y \in I \text{ and } y < 0$

$shift2_I : I \times I \to I \cup \{\textbf{integer\_overflow}, \textbf{invalid}\}$

$shift2_I(x, y) \quad = result_I(\lfloor x * 2^y \rfloor) \qquad \text{if } x, y \in I$

$shift10_I : I \times I \to I \cup \{\textbf{integer\_overflow}, \textbf{invalid}\}$

$shift10_I(x, y) \quad = result_I(\lfloor x * 10^y \rfloor) \qquad \text{if } x, y \in I$

5.1.5 Integer square root (rounded to nearest integer) operation

$sqrt_I : I \to I \cup \{\textbf{invalid}\}$

$sqrt_I(x) \quad = \text{round}(\sqrt{x}) \qquad \text{if } x \in I \text{ and } x \geq 0$

$= \textbf{invalid} \qquad \text{if } x \in I \text{ and } x < 0$

5.1.6 Divisibility and even/odd test operations

$divides_I : I \times I \rightarrow Boolean$

$$
\begin{array}{lll}
divides_I(x,y) & = \textbf{true} & \text{if } x,y \in I \text{ and } x|y \\
 & = \textbf{false} & \text{if } x,y \in I \text{ and not } x|y
\end{array}
$$

NOTES

1 $divides_I(0,0) = \textbf{false}$, since 0 does not divide anything, not even 0.

2 $divides_I$ cannot be implemented as, e.g., $eq_I(0, rem^f_I(y,x))$, since the remainder functions are **undefined** for a zero second argument.

$even_I : I \rightarrow Boolean$

$$
\begin{array}{lll}
even_I(x) & = \textbf{true} & \text{if } x \in I \text{ and } 2|x \\
 & = \textbf{false} & \text{if } x \in I \text{ and not } 2|x
\end{array}
$$

$odd_I : I \rightarrow Boolean$

$$
\begin{array}{lll}
odd_I(x) & = \textbf{true} & \text{if } x \in I \text{ and not } 2|x \\
 & = \textbf{false} & \text{if } x \in I \text{ and } 2|x
\end{array}
$$

5.1.7 Additional integer division and remainder operations

$quot_I : I \times I \rightarrow I \cup \{\textbf{integer\_overflow}, \textbf{invalid}\}$

$$
\begin{array}{lll}
quot_I(x,y) & = result_I(\lceil x/y \rceil) & \text{if } x,y \in I \text{ and } y \neq 0 \\
 & = \textbf{invalid} & \text{if } x \in I \text{ and } y = 0
\end{array}
$$

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

$$
\begin{array}{lll}
pad_I(x,y) & = (\lceil x/y \rceil * y) - x & \text{if } x,y \in I \text{ and } y \neq 0 \\
 & = \textbf{invalid} & \text{if } x \in I \text{ and } y = 0
\end{array}
$$

$remc_I : I \times I \rightarrow I \cup \{\textbf{integer\_overflow}, \textbf{invalid}\}$

$$
\begin{array}{lll}
remc_I(x,y) & = result_I(x - (\lceil x/y \rceil * y)) & \text{if } x,y \in I \text{ and } y \neq 0 \\
 & = \textbf{invalid} & \text{if } x \in I \text{ and } y = 0
\end{array}
$$

$divr_I : I \times I \rightarrow I \cup \{\textbf{integer\_overflow}, \textbf{invalid}\}$

$$
\begin{array}{lll}
divr_I(x,y) & = result_I(\text{round}(x/y)) & \text{if } x,y \in I \text{ and } y \neq 0 \\
 & = \textbf{invalid} & \text{if } x \in I \text{ and } y = 0
\end{array}
$$

$remr_I : I \times I \rightarrow I \cup \{\textbf{integer\_overflow}, \textbf{invalid}\}$

$$
\begin{array}{lll}
remr_I(x,y) & = result_I(x - (\text{round}(x/y) * y)) & \\
 & & \text{if } x,y \in I \text{ and } y \neq 0 \\
 & = \textbf{invalid} & \text{if } x \in I \text{ and } y = 0
\end{array}
$$

NOTE – $remc_I$ and $remr_I$ can overflow only for unsigned integer datatypes ($min_I = 0$).

5.1.8 Greatest common divisor and least common multiple operations

$gcd_I : I \times I \to I \cup \{\textbf{integer\_overflow}, \textbf{invalid}\}$

$$
\begin{aligned}
gcd_I(x,y) &= result_I(\max\{v \in \mathcal{Z} \mid v|x \text{ and } v|y\}) \\
&\qquad \text{if } x, y \in I \text{ and } (x \neq 0 \text{ or } y \neq 0) \\
&= \textbf{invalid} \qquad \text{if } x = 0 \text{ and } y = 0 \text{ and } +\infty \text{ is not available}
\end{aligned}
$$

NOTES

1 Returning 0 for $gcd_I(0,0)$, as is sometimes suggested, would be incorrect, since the greatest common divisor for 0 and 0 is infinity.

2 gcd_I will overflow only if $bounded_I = \textbf{true}$, $minint_I = -maxint_I - 1$, and both arguments to gcd_I are $minint_I$. The greatest common divisor is then $-minint_I$, which is then not in I.

$lcm_I : I \times I \to I \cup \{\textbf{integer\_overflow}\}$

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

NOTE 3 – $lcm_I(x,y)$ overflows for many arguments: e.g., if x and y are relative primes, then the least common multiple is $|x * y|$, which may be greater than $maxint_I$.

$gcd\_seq_I : [I] \to I \cup \{\textbf{integer\_overflow}, \textbf{invalid}\}$

$$
\begin{aligned}
&gcd\_seq_I([x_1, ..., x_n]) \\
&\qquad = result_I(\max\{v \in \mathcal{Z} \mid v|x_i \text{ for all } i \in \{1, ..., n\}\}) \\
&\qquad\qquad \text{if } \{x_1, ..., x_n\} \subset I \text{ and } \{0\} \neq \{x_1, ..., x_n\} \\
&\qquad = \textbf{invalid} \qquad \text{if } \{0\} = \{x_1, ..., x_n\} \text{ and } +\infty \text{ is not available}
\end{aligned}
$$

$lcm\_seq_I : [I] \to I \cup \{\textbf{integer\_overflow}\}$

$$
\begin{aligned}
&lcm\_seq_I([x_1, ..., x_n]) \\
&\qquad = result_I(\min\{v \in \mathcal{Z} \mid x_i|v \text{ for all } i \in \{1, ..., n\} \text{ and } v > 0\}) \\
&\qquad\qquad \text{if } \{x_1, ..., x_n\} \subset I \text{ and } 0 \notin \{x_1, ..., x_n\} \\
&\qquad = 0 \qquad \text{if } \{x_1, ..., x_n\} \subset I \text{ and } 0 \in \{x_1, ..., x_n\}
\end{aligned}
$$

5.1.9 Support operations for extended integer range

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

$add\_wrap_I : I \times I \to I$

$add\_wrap_I(x,y) = wrap_I(x + y) \qquad \text{if } x, y \in I$

$add\_ov_I : I \times I \to \{-1, 0, 1\}$

$$
\begin{aligned}
add\_ov_I(x,y) &= ((x + y) - add\_wrap_I(x,y))/(maxint_I - minint_I + 1) \\
&\qquad \text{if } x, y \in I \text{ and } I \neq \mathcal{Z} \\
&= 0 \qquad \text{if } x, y \in I \text{ and } I = \mathcal{Z}
\end{aligned}
$$

$sub\_wrap_I : I \times I \to I$

$$sub\_wrap_I(x,y) = wrap_I(x-y) \qquad \text{if } x,y \in I$$

$$sub\_ov_I : I \times I \to \{-1,0,1\}$$
$$sub\_ov_I(x,y) = ((x-y) - sub\_wrap_I(x,y))/(maxint_I - minint_I + 1) \qquad \text{if } x,y \in I \text{ and } I \neq \mathcal{Z}$$
$$= 0 \qquad \text{if } x,y \in I \text{ and } I = \mathcal{Z}$$

$$mul\_wrap_I : I \times I \to I$$
$$mul\_wrap_I(x,y) = wrap_I(x * y) \qquad \text{if } x,y \in I$$

$$mul\_ov_I : I \times I \to I$$
$$mul\_ov_I(x,y) = ((x * y) - mul\_wrap_I(x,y))/(maxint_I - minint_I + 1) \qquad \text{if } x,y \in I \text{ and } I \neq \mathcal{Z}$$
$$= 0 \qquad \text{if } x,y \in I \text{ and } I = \mathcal{Z}$$

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 Additional basic floating point operations

Clause 5.2 of ISO/IEC 10967-1 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 a floating point type conforming to ISO/IEC 10967-1. Floating point datatypes conforming to ISO/IEC 10967-1 usually do contain **−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:

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

is a rounding function. It rounds towards negative infinity.

NOTE 1 – F^* is defined in ISO/IEC 10967-1. It is the unbounded extension of F.

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

is a rounding function. It rounds towards positive infinity.

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

is a rounding function, that is partially implementation defined. It rounds to nearest. The handling of ties is implementation defined, but must be sign symmetric. If $iec\_559_F =$ **true**, the semantics of $nearest_F$ is completely determined: ties are rounded to even last digit by $nearest_F$.

$result_F$ is a helper function that is partially implementation defined. The specification from ISO/IEC 10967-1 is repeated here, but here details regarding continuation values upon overflow and underflow are given.

NOTE 2 – These details are intended to be in accordance with IEC 559 when $iec\_559_F =$ **true**.

$result_F : \mathcal{R} \times (\mathcal{R} \to F^*) \to F \cup \{\textbf{underflow}, \textbf{floating\_overflow}\}$

$result_F(x, nearest_F) = \textbf{floating\_overflow}(+\infty)$ if $nearest_F(x) > fmax_F$
$result_F(x, nearest_F) = \textbf{floating\_overflow}(-\infty)$ if $nearest_F(x) < -fmax_F$
$result_F(x, up_F) = \textbf{floating\_overflow}(+\infty)$ if $up_F(x) > fmax_F$
$result_F(x, up_F) = \textbf{floating\_overflow}(-fmax_F)$ if $up_F(x) < -fmax_F$
$result_F(x, down_F) = \textbf{floating\_overflow}(fmax_F)$ if $down_F(x) > fmax_F$
$result_F(x, down_F) = \textbf{floating\_overflow}(-\infty)$ if $down_F(x) < -fmax_F$

$result_F(x, rnd) = rnd(x)$ if $fminN_F \le |x|$ and $|rnd(x)| \le fmax_F$
$= 0$ if $x = 0$
$= \textbf{underflow}(rnd(x))$ if $denorm_F = \textbf{true}$ and $(rnd(x) < 0$ or $x > 0)$ and $|x| < fminN_F$ and $x \ne rnd(x)$
$= x$ if $iec\_559_F = \textbf{true}$ and $x \ne 0$ and $|x| < fminN_F$ and $x = rnd(x)$ and underflow is only recorded in indicator
$= \textbf{underflow}(x)$ if $iec\_559_F = \textbf{true}$ and $x \ne 0$ and $|x| < fminN_F$ and $x = rnd(x)$ and underflow is trapped
$= rnd(x)$ or $\textbf{underflow}(rnd(x))$ if $iec\_559_F = \textbf{false}$ and $denorm_F = \textbf{true}$ and $x \ne 0$ and $|x| < fminN_F$ and $x = rnd(x)$
$= \textbf{underflow}(\textbf{-0})$ if $denorm_F = \textbf{true}$ and **-0** is available and $rnd(x) = 0$ and $x < 0$
$= \textbf{underflow}(0)$ if $denorm_F = \textbf{true}$ and **-0** is not available and $rnd(x) = 0$ and $x < 0$
$= \textbf{underflow}(0)$ if $denorm_F = \textbf{false}$ and $0 < x$ and $x < fminN_F$
$= \textbf{underflow}(\textbf{-0})$ if $denorm_F = \textbf{false}$ and **-0** is available and $-fminN_F < x$ and $x < 0$
$= \textbf{underflow}(0)$ if $denorm_F = \textbf{false}$ and **-0** is not available and $-fminN_F < x$ and $x < 0$

NOTE 3 – $denorm_F = \textbf{false}$ implies $iec\_559_F = \textbf{false}$, and $iec\_559_F = \textbf{true}$ implies $denorm_F = \textbf{true}$.

5.2.2 Floating point maximum and minimum operations

What the maximum and minimum operations should return on one quiet **NaN** (**qNaN**) input depends on the context. 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, if both are included in the ISO/IEC 10967-2 binding.

$max_F : F \times F \to F$

$max_F(x, y) = \max\{x, y\}$ if $x, y \in F$
$= +\infty$ if $x = +\infty$ and $y \in F \cup \{-\infty, \textbf{-0}\}$
$= y$ if $x = \textbf{-0}$ and $y \in F$ and $y \ge 0$
$= \textbf{-0}$ if $x = \textbf{-0}$ and $((y \in F$ and $y < 0)$ or $y = \textbf{-0})$
$= y$ if $x = -\infty$ and $y \in F \cup \{+\infty, \textbf{-0}\}$

	$= +\infty$	if $y = +\infty$ and $x \in F \cup \{+\infty, \mathbf{-0}\}$
	$= x$	if $y = \mathbf{-0}$ and $x \in F$ and $x \geq 0$
	$= \mathbf{-0}$	if $y = \mathbf{-0}$ and $x \in F$ and $x < 0$
	$= x$	if $y = -\infty$ and $x \in F \cup \{-\infty, \mathbf{-0}\}$
	$= \mathbf{qNaN}$	if x is a quiet NaN and y is not a signalling NaN
	$= \mathbf{qNaN}$	if y is a quiet NaN and x is not a signalling NaN
	$= \mathbf{invalid}(\mathbf{qNaN})$	if x is a signalling NaN or y is a signalling NaN

$min_F : F \times F \to F$

$min_F(x, y)$	$= \min\{x, y\}$	if $x, y \in F$
	$= y$	if $x = +\infty$ and $y \in F \cup \{-\infty, \mathbf{-0}\}$
	$= \mathbf{-0}$	if $x = \mathbf{-0}$ and $y \in F$ and $y \geq 0$
	$= y$	if $x = \mathbf{-0}$ and $((y \in F$ and $y < 0)$ or $y = \mathbf{-0})$
	$= -\infty$	if $x = -\infty$ and $y \in F \cup \{+\infty, \mathbf{-0}\}$
	$= x$	if $y = +\infty$ and $x \in F \cup \{+\infty, \mathbf{-0}\}$
	$= \mathbf{-0}$	if $y = \mathbf{-0}$ and $x \in F$ and $x \geq 0$
	$= x$	if $y = \mathbf{-0}$ and $x \in F$ and $x < 0$
	$= -\infty$	if $y = -\infty$ and $x \in F \cup \{-\infty, \mathbf{-0}\}$
	$= \mathbf{qNaN}$	if x is a quiet NaN and y is not a signalling NaN
	$= \mathbf{qNaN}$	if y is a quiet NaN and x is not a signalling NaN
	$= \mathbf{invalid}(\mathbf{qNaN})$	if x is a signalling NaN or y is a signalling NaN

$mmax_F : F \times F \to F$

$mmax_F(x, y)$	$= max_F(x, y)$	if $x, y \in F \cup \{+\infty, \mathbf{-0}, -\infty\}$
	$= x$	if $x \in F \cup \{+\infty, \mathbf{-0}, -\infty\}$ and y is a quiet NaN
	$= y$	if $y \in F \cup \{+\infty, \mathbf{-0}, -\infty\}$ and x is a quiet NaN
	$= \mathbf{qNaN}$	if x is a quiet NaN and y is a quiet NaN
	$= \mathbf{invalid}(\mathbf{qNaN})$	if x is a signalling NaN or y is a signalling NaN

$mmin_F : F \times F \to F$

$mmin_F(x, y)$	$= min_F(x, y)$	if $x, y \in F \cup \{+\infty, \mathbf{-0}, -\infty\}$
	$= x$	if $x \in F \cup \{+\infty, \mathbf{-0}, -\infty\}$ and y is a quiet NaN
	$= y$	if $y \in F \cup \{+\infty, \mathbf{-0}, -\infty\}$ and x is a quiet NaN
	$= \mathbf{qNaN}$	if x is a quiet NaN and y is a quiet NaN
	$= \mathbf{invalid}(\mathbf{qNaN})$	if x is a signalling NaN or y is a signalling NaN

NOTE – If one of the arguments to $mmax_F$ or $mmin_F$ is a quiet NaN, that argument is ignored.

$max\_seq_F : [F] \to F \cup \{-\infty, \mathbf{invalid}\}$

$max\_seq_F([x_1, ..., x_n])$

	$= -\infty$	if $n = 0$ and $-\infty$ is available
	$= \mathbf{invalid}(\mathbf{qNaN})$	if $n = 0$ and $-\infty$ is not available
	$= x_1$	if $n = 1$ and x_1 is not a NaN
	$= \mathbf{qNaN}$	if $n = 1$ and x_1 is a quiet NaN
	$= \mathbf{invalid}(\mathbf{qNaN})$	if $n = 1$ and x_1 is a signalling NaN
	$= max_F(max\_seq_F([x_1, ..., x_{n-1}]), x_n)$	

$$\text{if } n \geq 2$$

$min\_seq_F : [F] \to F \cup \{+\infty, \textbf{invalid}\}$

$min\_seq_F([x_1, ..., x_n])$

$= +\infty$ if $n = 0$ and $+\infty$ is available

$= \textbf{invalid}(\textbf{qNaN})$ if $n = 0$ and $+\infty$ is not available

$= x_1$ if $n = 1$ and x_1 is not a NaN

$= \textbf{qNaN}$ if $n = 1$ and x_1 is a quiet NaN

$= \textbf{invalid}(\textbf{qNaN})$ if $n = 1$ and x_1 is a signalling NaN

$= min_F(min\_seq_F([x_1, ..., x_{n-1}]), x_n)$ if $n \geq 2$

$mmax\_seq_F : [F] \to F \cup \{-\infty, \textbf{invalid}\}$

$mmax\_seq_F([x_1, ..., x_n])$

$= -\infty$ if $n = 0$ and $-\infty$ is available

$= \textbf{invalid}(\textbf{qNaN})$ if $n = 0$ and $-\infty$ is not available

$= x_1$ if $n = 1$ and x_1 is not a signalling NaN

$= \textbf{invalid}(\textbf{qNaN})$ if $n = 1$ and x_1 is a signalling NaN

$= mmax_F(mmax\_seq_F([x_1, ..., x_{n-1}]), x_n)$ if $n \geq 2$

$mmin\_seq_F : [F] \to F \cup \{+\infty, \textbf{invalid}\}$

$mmin\_seq_F([x_1, ..., x_n])$

$= +\infty$ if $n = 0$ and $+\infty$ is available

$= \textbf{invalid}(\textbf{qNaN})$ if $n = 0$ and $+\infty$ is not available

$= x_1$ if $n = 1$ and x_1 is not a signalling NaN

$= \textbf{invalid}(\textbf{qNaN})$ if $n = 1$ and x_1 is a signalling NaN

$= mmin_F(mmin\_seq_F([x_1, ..., x_{n-1}]), x_n)$ if $n \geq 2$

5.2.3 Floating point positive difference (monus, diminish) operation

$dim_F : F \times F \to F \cup \{\textbf{floating\_overflow}, \textbf{underflow}\}$

$dim_F(x, y) = result_F(\max\{0, x - y)\}, rnd_F)$ if $x, y \in F$

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

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

$= +\infty$ if $x = +\infty$ and $y \in F \cup \{-\infty\}$

$= \textbf{invalid}(\textbf{qNaN})$ if $x = +\infty$ and $y = +\infty$

$= 0$ if $x = -\infty$ and $y \in F \cup \{+\infty\}$

$= \textbf{invalid}(\textbf{qNaN})$ if $x = -\infty$ and $y = -\infty$

$= 0$ if $y = +\infty$ and $x \in F$

$= +\infty$ if $y = -\infty$ and $x \in F$

$= \textbf{qNaN}$ if x is a quiet NaN and y is not a signalling NaN

$= \textbf{qNaN}$ if y is a quiet NaN and x is not a signalling NaN

$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN

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 Round, floor, and ceiling operations

$rounding_F : F \to F \cup \{-\mathbf{0}\}$

$$
\begin{array}{lll}
rounding_F(x) & = \mathrm{round}(x) & \text{if } x \in F \text{ and } (x \geq 0 \text{ or } \mathrm{round}(x) \neq 0) \\
& = neg_F(0) & \text{if } x \in F \text{ and } x < 0 \text{ and } \mathrm{round}(x) = 0 \\
& = -\mathbf{0} & \text{if } x = -\mathbf{0} \\
& = +\infty & \text{if } x = +\infty \\
& = -\infty & \text{if } x = -\infty \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

$floor_F : F \to F$

$$
\begin{array}{lll}
floor_F(x) & = \lfloor x \rfloor & \text{if } x \in F \\
& = -\mathbf{0} & \text{if } x = -\mathbf{0} \\
& = +\infty & \text{if } x = +\infty \\
& = -\infty & \text{if } x = -\infty \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

$ceiling_F : F \to F \cup \{-\mathbf{0}\}$

$$
\begin{array}{lll}
ceiling_F(x) & = \lceil x \rceil & \text{if } x \in F \text{ and } (x \geq 0 \text{ or } \lceil x \rceil \neq 0) \\
& = neg_F(0) & \text{if } x \in F \text{ and } x < 0 \text{ and } \lceil x \rceil = 0 \\
& = -\mathbf{0} & \text{if } x = -\mathbf{0} \\
& = +\infty & \text{if } x = +\infty \\
& = -\infty & \text{if } x = -\infty \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTES

1 The result in the second case for $rounding_F$ and $ceiling_F$ is 0, if $-\mathbf{0}$ is not in the type corresponding to F, otherwise it is $-\mathbf{0}$.

2 $floor_F(x) = neg_F(ceiling_F(neg_F(x)))$,
$ceiling_F(x) = neg_F(floor_F(neg_F(x)))$, and
$rounding_F(x) = neg_F(rounding_F(neg_F(x)))$.
Negative zeroes, if available, are handed in such a way as to maintain these identites.

3 Truncate to integer is specified in ISO/IEC 10967-1:1994, by the name $intpart_F$.

$rounding\_rest_F : F \to F$

$$
\begin{array}{lll}
rounding\_rest_F(x) & & \\
& = x - \mathrm{round}(x) & \text{if } x \in F \\
& = 0 & \text{if } x = -\mathbf{0} \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x = +\infty \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x = -\infty \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

$floor\_rest_F : F \to F$

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

$$
ceiling\_rest_F : F \to F
$$

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

NOTE 4 – The rest after truncation is specified in ISO/IEC 10967-1:1994, by the name $fractpart_F$.

5.2.5 Operation for remainder after division and round to integer (IEEE remainder)

$$
irem_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{invalid}\}
$$

$$
\begin{aligned}
irem_F(x,y) &= result_F(x - (\text{round}(x/y) * y), nearest_F) \\
&\qquad \text{if } x, y \in F \text{ and } y \neq 0 \text{ and} \\
&\qquad\quad (x \geq 0 \text{ or } x - (\text{round}(x/y) * y) \neq 0) \\
&= \mathbf{-0} && \text{if } x, y \in F \text{ and } y \neq 0 \text{ and} \\
&\qquad\quad x < 0 \text{ and } x - (\text{round}(x/y) * y) = 0 \\
&= \mathbf{-0} && \text{if } x = \mathbf{-0} \text{ and } y \in F \cup \{-\infty, +\infty\} \text{ and } y \neq 0 \\
&= x && \text{if } x \in F \text{ and } y \in \{-\infty, +\infty\} \\
&= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \in F \cup \{-\infty, \mathbf{-0}, +\infty\} \text{ and } y = \mathbf{-0} \\
&= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \in F \cup \{\mathbf{-0}\} \text{ and } y = 0 \\
&= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \in \{-\infty, +\infty\} \text{ and } y \in F \cup \{-\infty, +\infty\} \\
&= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN and } y \text{is not a signalling NaN} \\
&= \mathbf{qNaN} && \text{if } y \text{ is a quiet NaN and } x \text{ is not a signalling NaN} \\
&= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN or } y \text{ is a signalling NaN}
\end{aligned}
$$

5.2.6 Square root and reciprocal square root operations

$$
sqrt_F : F \to F \cup \{\mathbf{invalid}\}
$$

$$
\begin{aligned}
sqrt_F(x) &= nearest_F(\sqrt{x}) && \text{if } x \in F \text{ and } x \geq 0 \\
&= \mathbf{-0} && \text{if } x = \mathbf{-0} \\
&= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } (x \in F \text{ and } x < 0) \text{ or } x = -\infty \\
&= +\infty && \text{if } x = +\infty \\
&= \mathbf{qNaN} && \text{if } x \text{ is a quiet NaN} \\
&= \mathbf{invalid}(\mathbf{qNaN}) && \text{if } x \text{ is a signalling NaN}
\end{aligned}
$$

$$
rec\_sqrt_F : F \to F \cup \{\mathbf{invalid}, \mathbf{pole}\}
$$

$$
\begin{array}{lll}
rec\_sqrt_F(x) & = rnd_F(1/\sqrt{x}) & \text{if } x \in F \text{ and } x > 0 \\
& = \mathbf{pole}(+\infty) & \text{if } x \in F \text{ and } x = 0 \\
& = \mathbf{pole}(+\infty) & \text{if } x = \mathbf{-0} \\
& = 0 & \text{if } x = +\infty \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } (x \in F \text{ and } x < 0) \text{ or } x = -\infty \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

5.2.7 Support operations for extended floating point precision

$$add\_lo_F : F \times F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$$

$$
\begin{array}{lll}
add\_lo_F(x,y) & = result_F((x+y) - rnd_F(x+y), rnd_F) & \\
& & \text{if } x, y, add_F(x,y) \in F \\
& = \mathbf{underflow}(0)? & \text{if } add_F(x,y) = \mathbf{underflow}(u) \\
& = 0? & \text{if } add_F(x,y) = \mathbf{floating\_overflow}(+\infty) \\
& = 0? & \text{if } add_F(x,y) = \mathbf{floating\_overflow}(-\infty) \\
& = add\_lo_F(0,y) & \text{if } x = \mathbf{-0} \text{ and } y \in F \cup \{-\infty, \mathbf{-0}, +\infty\} \\
& = add\_lo_F(x,0) & \text{if } y = \mathbf{-0} \text{ and } x \in F \cup \{-\infty, +\infty\} \\
& = \mathbf{invalid}(\mathbf{qNaN})? & \text{if } x \in \{-\infty, +\infty\} \text{ and } y \in F \cup \{-\infty, +\infty\} \\
& = \mathbf{invalid}(\mathbf{qNaN})? & \text{if } y \in \{-\infty, +\infty\} \text{ and } x \in F \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN and } y \text{ is not a signalling NaN} \\
& = \mathbf{qNaN} & \text{if } y \text{ is a quiet NaN and } x \text{ is not a signalling NaN} \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN or } y \text{ is a signalling NaN}
\end{array}
$$

$$sub\_lo_F : F \times F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$$

$$
\begin{array}{lll}
sub\_lo_F(x,y) & = result_F((x-y) - rnd_F(x-y), rnd_F) & \\
& & \text{if } x, y, sub_F(x,y) \in F \\
& = \mathbf{underflow}(0)? & \text{if } sub_F(x,y) = \mathbf{underflow}(u) \\
& = \mathbf{floating\_overflow}(-\infty)?0? & \\
& & \text{if } sub_F(x,y) = \mathbf{floating\_overflow}(+\infty) \\
& = \mathbf{floating\_overflow}(+\infty)?0? & \\
& & \text{if } sub_F(x,y) = \mathbf{floating\_overflow}(-\infty) \\
& = sub\_lo_F(0,y) & \text{if } x = \mathbf{-0} \text{ and } y \in F \cup \{-\infty, \mathbf{-0}, +\infty\} \\
& = sub\_lo_F(x,0) & \text{if } y = \mathbf{-0} \text{ and } x \in F \cup \{-\infty, +\infty\} \\
& = \mathbf{invalid}(\mathbf{qNaN})? & \text{if } x \in \{-\infty, +\infty\} \text{ and } y \in F \cup \{-\infty, +\infty\} \\
& = \mathbf{invalid}(\mathbf{qNaN})? & \text{if } y \in \{-\infty, +\infty\} \text{ and } x \in F \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN and } y \text{ is not a signalling NaN} \\
& = \mathbf{qNaN} & \text{if } y \text{ is a quiet NaN and } x \text{ is not a signalling NaN} \\
& = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN or } y \text{ is a signalling NaN}
\end{array}
$$

NOTES

1 If $rnd\_style_F = nearest$, then, in the absence of notifications, add_lo_F and sub_lo_F returns exact results.

2 $sub\_lo_F(x,y) = add\_lo_F(x, neg_F(y))$.

$$mul\_lo_F : F \times F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$$

$$
\begin{array}{lll}
mul\_lo_F(x,y) & = result_F((x*y) - rnd_F(x*y), rnd_F) & \\
& & \text{if } x, y, mul_F(x,y) \in F \\
& = \mathbf{underflow}(0)? & \text{if } mul_F(x,y) = \mathbf{underflow}(u)
\end{array}
$$

$= 0$ if $x, y \in F$ and $mul_F(x, y) = \mathbf{-0}$
$= \mathbf{floating\_overflow}(-\infty)?0?$ if $mul_F(x, y) = \mathbf{floating\_overflow}(+\infty)$
$= \mathbf{floating\_overflow}(+\infty)?0?$ if $mul_F(x, y) = \mathbf{floating\_overflow}(-\infty)$
$= mul\_lo_F(0, y)$ if $x = \mathbf{-0}$ and $y \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= mul\_lo_F(x, 0)$ if $y = \mathbf{-0}$ and $x \in F \cup \{-\infty, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})?$ if $x \in \{-\infty, +\infty\}$ and $y \in F \cup \{-\infty, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})?$ if $y \in \{-\infty, +\infty\}$ and $x \in F$
$= \mathbf{qNaN}$ if x is a quiet NaN and y is not a signalling NaN
$= \mathbf{qNaN}$ if y is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN

NOTE 3 – In the absence of notifications, mul_lo_F returns an exact result.

$div\_rest_F : F \times F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}, \mathbf{invalid}\}$

$div\_rest_F(x, y) = result_F(x - (y * rnd_F(x/y)), rnd_F)$ if $x, y, div_F(x, y) \in F$
$= result_F(x - (y * u), rnd_F)$ if $div_F(x, y) = \mathbf{underflow}(u)$ and $z \in F$
$= x$ if $x, y \in F$ and $(div_F(x, y) = \mathbf{-0}$ or $div_F(x, y) = \mathbf{underflow}(\mathbf{-0}))$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $x \in F$ and $y = 0$
$= \mathbf{floating\_overflow}(-\infty)?0?$ if $div_F(x, y) = \mathbf{floating\_overflow}(+\infty)$
$= \mathbf{floating\_overflow}(+\infty)?0?$ if $div_F(x, y) = \mathbf{floating\_overflow}(-\infty)$
$= div\_rest_F(0, y)$ if $x = \mathbf{-0}$ and $y \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $y = \mathbf{-0}$ and $x \in F \cup \{-\infty, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})?$ if $x \in \{-\infty, +\infty\}$ and $y \in F \cup \{-\infty, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})?$ if $y \in \{-\infty, +\infty\}$ and $x \in F$
$= \mathbf{qNaN}$ if x is a quiet NaN and y is not a signalling NaN
$= \mathbf{qNaN}$ if y is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN

$sqrt\_rest_F : F \to F \cup \{\mathbf{underflow}, \mathbf{invalid}\}$

$sqrt\_rest_F(x) = result_F(x - (sqrt_F(x) * sqrt_F(x)), rnd_F)$ if $x \in F$ and $x \geq 0$
$= \mathbf{-0}$ if $x = \mathbf{-0}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $(x \in F$ and $x < 0)$ or $x = -\infty$
$= \mathbf{invalid}(\mathbf{qNaN})?0?$ if $x = +\infty$
$= \mathbf{qNaN}$ if x is a quiet NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN

NOTE 4 – $sqrt\_rest_F(x)$ is exact when there is no **underflow**.

$add3_F : F \times F \times F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$

$add3_F(x, y, z) = result_F((x + y) + z, rnd_F)$ if $x, y, z \in F$

$= add_F(add_F(x,y),z)$ if $x \in \{-\infty, \mathbf{-0}, +\infty\}$ and $y,z \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= add_F(add_F(x,y),z)$ if $y \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F$ and $z \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= add_F(add_F(x,y),z)$ if $z \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x,y \in F$
$= \mathbf{qNaN}$ if x is a quiet NaN and not y nor z is a signalling NaN
$= \mathbf{qNaN}$ if y is a quiet NaN and not x nor z is a signalling NaN
$= \mathbf{qNaN}$ if z is a quiet NaN and not x nor y is a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN or z is a signalling NaN

NOTE 5 – $add3_F(x,y,z) = add_F(add_F(x,y),z)$ if $x \notin F$ or $y \notin F$ or $z \notin F$, thus $add3_F(\mathbf{-0},\mathbf{-0},\mathbf{-0}) = \mathbf{-0}$.

$add3\_mid_F : F \times F \times F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$

$add3\_mid_F(x,y,z)$
$= result_F(((x+y)+z) - rnd_F((x+y)+z), rnd_F)$ if $x,y,z, add3_F(x,y,z) \in F$
$= \mathbf{underflow}(0)?$ if $add3_F(x,y,z) = \mathbf{underflow}(u)$
$= \mathbf{floating\_overflow}(-\infty)?0?$ if $add3_F(x,y,z) = \mathbf{floating\_overflow}(+\infty)$
$= \mathbf{floating\_overflow}(+\infty)?0?$ if $add3_F(x,y,z) = \mathbf{floating\_overflow}(-\infty)$
$= add3\_mid_F(0,y,z)$ if $x = \mathbf{-0}$ and $y,z \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= add3\_mid_F(x,0,z)$ if $y = \mathbf{-0}$ and $x \in F \cup \{-\infty, +\infty\}$ and $z \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= add3\_mid_F(x,y,0)$ if $z = \mathbf{-0}$ and $x,y \in F \cup \{-\infty, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})?$ if $x \in \{-\infty, +\infty\}$ and $y,z \in F \cup \{-\infty, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})?$ if $y \in \{-\infty, +\infty\}$ and $x \in F$ and $z \in F \cup \{-\infty, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})?$ if $z \in \{-\infty, +\infty\}$ and $x,y \in F$
$= \mathbf{qNaN}$ if x is a quiet NaN and not y nor z is a signalling NaN
$= \mathbf{qNaN}$ if y is a quiet NaN and not x nor z is a signalling NaN
$= \mathbf{qNaN}$ if z is a quiet NaN and not x nor y is a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN or z is a signalling NaN

$mul\_add_F : F \times F \times F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$

$mul\_add_F(x,y,z)$
$= result_F((x * y) + z, rnd_F)$ if $x,y,z \in F$ and $x \neq 0$ and $y \neq 0$ and $z \neq 0$
$= add_F(mul_F(x,y),z)$ if $x \in \{-\infty, \mathbf{-0}, 0, +\infty\}$ and $y,z \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= add_F(mul_F(x,y),z)$ if $y \in \{-\infty, \mathbf{-0}, 0, +\infty\}$ and $x \in F$ and $x \neq 0$ and $z \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= add_F(mul_F(x,y),z)$ if $z \in \{-\infty, \mathbf{-0}, 0, +\infty\}$ and $x, y \in F$ and $x \neq 0$ and $y \neq 0$

$= \mathbf{qNaN}$ if x is a quiet NaN and not y nor z is a signalling NaN

$= \mathbf{qNaN}$ if y is a quiet NaN and not x nor z is a signalling NaN

$= \mathbf{qNaN}$ if z is a quiet NaN and not x nor y is a signalling NaN

$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN or z is a signalling NaN

NOTE 6 – $mul\_add_F(x,y,z) = add_F(mul_F(x,y),z)$ if $x \notin F$ or $y \notin F$ or $z \notin F$ or $x = 0$ or $y = 0$ or $z = 0$. E.g., $mul\_add_F(\mathbf{-0}, 1, \mathbf{-0}) = \mathbf{-0}$.

$mul\_add\_mid_F : F \times F \times F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$

$mul\_add\_mid_F(x,y,z)$

$= result_F(((x * y) + z) - mul\_add_F(x,y,z), rnd_F)$ if $x, y, z, mul\_add_F(x,y,z) \in F$

$= \mathbf{underflow}(0)?$ if $mul\_add_F(x,y,z) = \mathbf{underflow}(u)$

$= \mathbf{floating\_overflow}(-\infty)?0?$ if $mul\_add_F(x,y,z) = \mathbf{floating\_overflow}(+\infty)$

$= \mathbf{floating\_overflow}(+\infty)?0?$ if $mul\_add_F(x,y,z) = \mathbf{floating\_overflow}(-\infty)$

$= mul\_add\_mid_F(0,y,z)$ if $x = \mathbf{-0}$ and $y, z \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= mul\_add\_mid_F(x,0,z)$ if $y = \mathbf{-0}$ and $x \in F \cup \{-\infty, +\infty\}$ and $z \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= mul\_add\_mid_F(x,y,0)$ if $z = \mathbf{-0}$ and $x, y \in F \cup \{-\infty, +\infty\}$

$= \mathbf{invalid}(\mathbf{qNaN})?$ if $x \in \{-\infty, +\infty\}$ and $y, z \in F \cup \{-\infty, +\infty\}$

$= \mathbf{invalid}(\mathbf{qNaN})?$ if $y \in \{-\infty, +\infty\}$ and $x \in F$ and $z \in F \cup \{-\infty, +\infty\}$

$= \mathbf{invalid}(\mathbf{qNaN})?$ if $z \in \{-\infty, +\infty\}$ and $x, y \in F$

$= \mathbf{qNaN}$ if x is a quiet NaN and not y nor z is a signalling NaN

$= \mathbf{qNaN}$ if y is a quiet NaN and not x nor z is a signalling NaN

$= \mathbf{qNaN}$ if z is a quiet NaN and not x nor y is a signalling NaN

$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN or z is a signalling NaN

For the following operation F' is a floating point type conforming to ISO/IEC 10967-1.

NOTE 7 – It is expected that $p_{F'} > p_F$, i.e. F' has higher precision than F, but that is not required.

$mul_{F \to F'} : F \times F \to F' \cup \{\mathbf{-0}, \mathbf{floating\_overflow}, \mathbf{underflow}\}$

$$
\begin{array}{lll}
mul_{F\to F'}(x,y) = result_{F'}(x * y, rnd_{F'}) & & \text{if } x,y \in F \text{ and } x \neq 0 \text{ and } y \neq 0 \\
= cvt_{F\to F'}(mul_F(x,y)) & & \text{if } x \in \{-\infty, \mathbf{-0}, 0, +\infty\} \text{ and} \\
 & & \quad y \in F \cup \{-\infty, \mathbf{-0}, +\infty\} \\
= cvt_{F\to F'}(mul_F(x,y)) & & \text{if } y \in \{-\infty, \mathbf{-0}, 0, +\infty\} \text{ and } x \in F \text{ and } x \neq 0 \\
= cvt_{F\to F'}(\mathbf{qNaN}) & & \text{if } x \text{ is a quiet NaN and } y \text{ is not a signalling NaN} \\
= cvt_{F\to F'}(\mathbf{qNaN}) & & \text{if } y \text{ is a quiet NaN and } x \text{ is not a signalling NaN} \\
= cvt_{F\to F'}(\mathbf{sNaN}) & & \text{if } x \text{ is a signalling NaN or } y \text{ is a signalling NaN}
\end{array}
$$

NOTE 8 – Converting a signalling **NaN** results in a notification of **invalid**. See clause 5.4.

5.2.8 Exact summation operation

An exact summation operation is useful for computing high accuracy sums, even if only the first element of the resulting list is ultimately kept.

In order to be able to specify the exact sum operation, which sums a sequence of floating point numbers returning a sequence of floating point numbers of decreasing magnitude, by p_F, a number of helper functions are needed.

The extended real addition helper function:

$$
\begin{array}{l}
add : (\mathcal{R} \cup \{-\infty, \mathbf{-0}, +\infty, \mathbf{qNaN}, \mathbf{sNaN}\}) \times (\mathcal{R} \cup \{-\infty, \mathbf{-0}, +\infty, \mathbf{qNaN}, \mathbf{sNaN}\}) \to \\
\quad (\mathcal{R} \cup \{-\infty, \mathbf{-0}, +\infty, \mathbf{qNaN}, \mathbf{sNaN}\})
\end{array}
$$

$$
\begin{array}{lll}
add(x,y) = x + y & & \text{if } x,y \in \mathcal{R} \\
= \mathbf{-0} & & \text{if } x = \mathbf{-0} \text{ and } y = \mathbf{-0} \\
= add(0,y) & & \text{if } x = \mathbf{-0} \text{ and } y \in \mathcal{R} \cup \{-\infty, +\infty\} \\
= add(x,0) & & \text{if } y = \mathbf{-0} \text{ and } x \in \mathcal{R} \cup \{-\infty, +\infty\} \\
= +\infty & & \text{if } x = +\infty \text{ and } y \in \mathcal{R} \cup \{+\infty\} \\
= +\infty & & \text{if } y = +\infty \text{ and } x \in \mathcal{R} \\
= -\infty & & \text{if } x = -\infty \text{ and } y \in \mathcal{R} \cup \{-\infty\} \\
= -\infty & & \text{if } y = -\infty \text{ and } x \in \mathcal{R} \\
= \mathbf{sNaN} & & \text{if } x = +\infty \text{ and } y = -\infty \\
= \mathbf{sNaN} & & \text{if } x = -\infty \text{ and } y = +\infty \\
= \mathbf{qNaN} & & \text{if } x \text{ is a quiet NaN and } y \text{ is not a signalling NaN} \\
= \mathbf{qNaN} & & \text{if } y \text{ is a quiet NaN and } x \text{ is not a signalling NaN} \\
= \mathbf{sNaN} & & \text{if } x \text{ is a signalling NaN or } y \text{ is a signalling NaN}
\end{array}
$$

The extended real summation helper function:

$$
sum : [\mathcal{R} \cup \{-\infty, \mathbf{-0}, +\infty, \mathbf{qNaN}, \mathbf{sNaN}\}] \to (\mathcal{R} \cup \{-\infty, \mathbf{-0}, +\infty, \mathbf{qNaN}, \mathbf{sNaN}\})
$$

$$
\begin{array}{lll}
sum([x_1, ..., x_n]) & & \\
= \mathbf{-0} & & \text{if } n = 0 \\
= add(sum([x_1, ..., x_{n-1}]), x_n) & & \\
 & & \text{if } n \geq 1
\end{array}
$$

The seq_result_F helper function:

$$
seq\_result_F : \mathcal{R} \times (\mathcal{R} \to F^*) \to [F] \cup \{\mathbf{floating\_overflow}\}
$$

$$
\begin{array}{lll}
seq\_result_F(x, rnd) & & \\
= [0] & & \text{if } x = 0 \text{ or } (x > 0 \text{ and } rnd(x) = 0 \text{ and } denorm_F = \mathbf{true}) \\
= [\mathbf{-0}] & & \text{if } x < 0 \text{ and } rnd(x) = 0 \text{ and } denorm_F = \mathbf{true} \\
= \mathbf{floating\_overflow}([+\infty]) & & \\
 & & \text{if } rnd(x) > fmax_F \\
= \mathbf{floating\_overflow}([-\infty]) & & \\
 & & \text{if } rnd(x) < -fmax_F
\end{array}
$$

$$= rnd(x) : seq\_result_F(x - rnd(x), rnd)$$
if $rnd(x) \neq 0$ and $rnd(x) \in F$ and $(denorm_F = \textbf{true}$ or $|x| \geq fminN_F)$

$$= [rnd(x - fminN_F), fminN_F]$$
if $-fminN_F < x$ and $x < 0$ and $denorm_F = \textbf{false}$

$$= [rnd(x + fminN_F), -fminN_F]$$
if $0 < x$ and $x < fminN_F$ and $denorm_F = \textbf{false}$

The exact summation operation:

$sum_F : [F] \rightarrow [F] \cup \{\textbf{floating\_overflow}\}$

$sum_F([x_1, ..., x_n])$

$= seq\_result_F(sum([x_1, ..., x_n]), nearest_F)$ if $sum([x_1, ..., x_n]) \in \mathcal{R}$ and $n \geq 1$

$= [sum([x_1, ..., x_n])]$ if $sum([x_1, ..., x_n]) \in \{-\infty, \textbf{-0}, +\infty\}$ and $n \geq 1$

$= [\textbf{-0}]$ if $n = 0$ and $\textbf{-0}$ is available

$= [0]$ if $n = 0$ and $\textbf{-0}$ is not available

$= [\textbf{qNaN}]$ if $sum([x_1, ..., x_n])$ is a quiet NaN

$= \textbf{invalid}([\textbf{qNaN}])$ if $sum([x_1, ..., x_n])$ is a signalling NaN

NOTE – $sum_F(sum_F(a)) = sum_F(a)$, and $sum_F(sum_F(a) + + sum_F(b)) = sum_F(a + + b)$ if there is no notification (where $++$ is sequence concatenation). Thus $sum_F([]) = sum_F([-\textbf{0}])$.

5.3 Elementary transcendental floating point operations

5.3.1 Specification format

5.3.1.1 Maximum error requirements

The specifications for each of the transcendental operations use an approximation helper function. The approximation helper functions are ideally identical to the true mathematical functions. However, that would imply that the maximum error for the corresponding operation was merely 0.5 ulp. This part of ISO/IEC 10967 does not require that the maximum error is only 0.5 ulp, but may 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 for the individual operations in this subclause have maximum error parameters that describe the maximum *relative* error of the helper function composed with $nearest_F$, for normalised results. The maximum error parameter also describe the maximum *absolute* error for subnormal continuation values if $denorm_F = \textbf{true}$. The relevant maximum error parameters shall be available to programs.

That for a helper function h_F, approximating f, the maximum error is $max\_error\_op_F$ means that for all arguments $x, ... \in F^* \times ...$ the following inequality is true:

$$|f(x, ...) - nearest_F(h_F(x, ...))| \leq max\_error\_op_F * r^{e_F(f(x,...))-p_F}$$

NOTES

1 Partially conforming implementations may have greater values for maximum error parameters than stipulated below. See annex B.

2 For most positive (and not too small) return values t, the true result is thus claimed to be in the interval $[t - (max\_error\_op_F * ulp_F(t)), t + (max\_error\_op_F * ulp_F(t))]$. But if the return value is exactly r_F^n for some $n \in \mathcal{Z}$, then the true result is claimed to be in the interval $[t - (max\_error\_op_F * ulp_F(t)/r_F), t + (max\_error\_op_F * ulp_F(t))]$, 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 be 0.5, the minimum value.

5.3.1.2 The *trans_result* helper function

The $trans\_result_F$ helper function is similar to the $result_F$ helper function extended with specifications for the continuation value on overflow, and it also returns **−0** for negative underflows that round (or are flushed) to zero, if possible. (Those extentions are implied in ISO/IEC 10967-1 for IEC 559 conforming implementations.) But $trans\_result_F$ is simplified compared to $result_F$ concerning **underflow**: $trans\_result_F$ always underflows for nonzero arguments that have an absolute value less than $fminN_F$, whereas $result_F$ does not always underflow then.

In addition, the rounding is fixed to $nearest_F$, rather than being parameterised. This is user visible only in the cases where the operation's approximation helper function is (required to be) exact, but where that value is not representable in F, e.g. e or π.

$$trans\_result_F : \mathcal{R} \to F \cup \{\textbf{underflow}, \textbf{floating\_overflow}\}$$

$trans\_result_F(x)$

$$\begin{aligned}
&= nearest_F(x) && \text{if } fminN_F \le |x| \text{ and } |nearest_F(x)| \le fmax_F \\
&= 0 && \text{if } x = 0 \\
&= \textbf{floating_overflow}(+\infty) \\
& && \text{if } nearest_F(x) > fmax_F \\
&= \textbf{floating_overflow}(-\infty) \\
& && \text{if } nearest_F(x) < -fmax_F \\
&= \textbf{underflow}(nearest_F(x)) \\
& && \text{if } denorm_F = \textbf{true} \text{ and } (nearest_F(x) < 0 \text{ or } x > 0) \\
& && \quad \text{and } |x| < fminN_F \\
&= \textbf{underflow}(\textbf{−0}) && \text{if } denorm_F = \textbf{true} \text{ and } \textbf{−0} \text{ is available and} \\
& && \quad nearest_F(x) = 0 \text{ and } x < 0 \\
&= \textbf{underflow}(0) && \text{if } denorm_F = \textbf{true} \text{ and } \textbf{−0} \text{ is not available and} \\
& && \quad nearest_F(x) = 0 \text{ and } x < 0 \\
&= \textbf{underflow}(0) && \text{if } denorm_F = \textbf{false} \text{ and} \\
& && \quad 0 < x \text{ and } x < fminN_F \\
&= \textbf{underflow}(\textbf{−0}) && \text{if } denorm_F = \textbf{false} \text{ and } \textbf{−0} \text{ is available and} \\
& && \quad -fminN_F < x \text{ and } x < 0 \\
&= \textbf{underflow}(0) && \text{if } denorm_F = \textbf{false} \text{ and } \textbf{−0} \text{ is not available and} \\
& && \quad -fminN_F < x \text{ and } x < 0
\end{aligned}$$

5.3.1.3 Sign requirements

The approximation helper functions are required to be zero exactly at the points where the approximated mathematical function is exactly zero. At points where the approximation helper functions are not zero, they are required to have the same sign as the approximated mathematical function at that point.

For the radian trigonometric helper functions, this sign requirement is imposed only for arguments, x, such that $|x| \le big\_angle\_r_F$ (see clause 5.3.6).

NOTE – For the operations, the continuation value after an **underflow** may be zero (or negative zero) as given by $trans\_result_F$, 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 559 infinities arguments. See the individual specifications.

5.3.1.4 Monotonicity requirements

When the maximum error is tight, i.e. 0.5 ulp, that implies that the approximation helper functions must be monotonous on the same intervals as the corresponding exact function is strictly monotonous. When the maximum error is greater than 0.5 ulp, and the rounding is not directed, a numerical function is not automatically monotonous where the corresponding exact function is.

The approximation helper functions in this clause are required to be monotonous on the same intervals as the mathematical functions they are approximating are monotonous. There is no general requirement that the approximation helper functions are strictly monotonous on the same intervals as the corresponding exact function is strictly monotonous, however, since such a requirement cannot be made due to that all floating point types are discrete, not continuous.

For the radian trigonometric helper functions, this monotonicity requirement is imposed only for arguments, x, such that $|x| \leq big\_angle\_r_F$ (see clause 5.3.6).

The unit argument trigonometric and unit argument inverse trigonometric approximating helper functions are excepted from the monotonicity requirement for the angular unit argument.

5.3.2 Hypotenuse operation

Maximum error parameter for the $hypot_F$ operation:

$max\_error\_hypot_F \in F$

The $max\_error\_hypot_F$ parameter is required to be in the interval $[0.5, 1]$.

The $hypot^*_F$ approximation helper function:

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

$hypot^*_F(x, y)$ 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:

$hypot^*_F(x, y) = hypot^*_F(y, x)$
$hypot^*_F(-x, y) = hypot^*_F(x, y)$
$hypot^*_F(x, y) \geq \max\{|x|, |y|\}$
$hypot^*_F(x, y) \leq |x| + |y|$
$hypot^*_F(x, y) \geq 1$ if $\sqrt{x^2 + y^2} \geq 1$
$hypot^*_F(x, y) \leq 1$ if $\sqrt{x^2 + y^2} \leq 1$

The $hypot_F$ operation:

$hypot_F : F \times F \to F \cup \{\textbf{underflow}, \textbf{floating\_overflow}\}$

$hypot_F(x, y) = trans\_result_F(hypot^*_F(x, y))$ if $x, y \in F$
$= hypot_F(0, y)$ if $x = \textbf{-0}$ and $y \in F \cup \{-\infty, \textbf{-0}, +\infty\}$
$= hypot_F(x, 0)$ if $y = \textbf{-0}$ and $x \in F \cup \{-\infty, +\infty\}$
$= +\infty$ if $x \in \{-\infty, +\infty\}$ and $y \in F \cup \{-\infty, +\infty\}$
$= +\infty$ if $y \in \{-\infty, +\infty\}$ and $x \in F$
$= \textbf{qNaN}$ if x is a quiet NaN and y is not a signalling NaN
$= \textbf{qNaN}$ if y is a quiet NaN and x is not a signalling NaN
$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN

5.3.3 Operations for exponentiations and logarithms

There are two maximum error parameters for approximate exponentiations and logarithms:

$max\_error\_exp_F \in F$
$max\_error\_power_F \in F$

The $max\_error\_exp_F$ parameter is required to be in the interval $[0.5, 1.5 * rnd\_error_F]$.

The $max\_error\_power_F$ parameter is required to be in the interval $[max\_error\_exp_F, 2 * rnd\_error_F]$.

NOTE 1 – The "exp" operations are thus required to be at least as accurate as the "power" operations.

e is the Napierian base.

NOTE 2 – $e = 2.71828....$ e is not in F.

5.3.3.1 Power-of e (natural exponentiation) operation

The exp^*_F approximation helper function:

$exp^*_F : F \to \mathcal{R}$

$exp^*_F(x)$ 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:

$exp^*_F(1) = e$
$exp^*_F(x) = 1$ if $x \in F$ and $exp^*_F(x) \neq e^x$ and $\ln(1 - (epsilon_F/(2 * r_F))) < x < \ln(1 + (epsilon_F/2))$

$exp^*_F(x) < fminD_F/2$ if $x \in F$ and $x < \ln(fminD_F) - 3$

The exp_F operation:

$exp_F : F \to F \cup \{\textbf{underflow}, \textbf{floating\_overflow}\}$

$$
\begin{array}{lll}
exp_F(x) & = trans\_result_F(exp^*_F(x)) & \text{if } x \in F \\
& = 1 & \text{if } x = \textbf{-0} \\
& = +\infty & \text{if } x = +\infty \\
& = 0 & \text{if } x = -\infty \\
& = \textbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \textbf{invalid}(\textbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTES

1 $exp_F(1) = nearest_F(e)$.

2 $exp_F(x)$ will overflow approximately when $x > \ln(fmax_F)$.

5.3.3.2 Operation for power-of e, minus one (natural exponentiation, minus one)

The $expm1^*_F$ approximation helper function:

$expm1^*_F : F \to \mathcal{R}$

$expm1^*_F(x)$ returns a close approximation to $e^x - 1$ in $\mathcal{R}$, with maximum error $max\_error\_exp_F$.

NOTE 1 – There are two advantages with the $expm1_F$ operation: Firstly, $expm1_F(x)$is much more accurate than $sub_F(exp_F(x), 1)$ when the exponent argument is close to zero. Secondly, the $expm1_F$ operation does not **underflow** for "very" negative exponent arguments. Something which may be advantageous if **underflow** handling is slow, and high accuracy for "very" negative arguments is not needed.

Further requirements on the $expm1^*_F$ approximation helper function:

$expm1^*_F(1) = e - 1$
$expm1^*_F(x) = x$ if $x \in F$ and $expm1^*_F(x) \neq e^x - 1$ and $-epsilon_F/r_F \leq x < 0.5 * epsilon_F/r_F$
$expm1^*_F(x) = -1$ if $x \in F$ and $expm1^*_F(x) \neq e^x - 1$ and $x < \ln(epsilon_F/(3 * r_F))$
$expm1^*_F(x) \leq exp^*_F(x)$ if $x \in F$

The $expm1_F$ operation:

$expm1_F : F \to F \cup \{\textbf{underflow}, \textbf{floating\_overflow}\}$

$expm1_F(x) = trans\_result_F(expm1^*_F(x))$ if $x \in F$ and $|x| \geq 0.5 * epsilon_F/r_F$
$= x$ if $x \in F$ and $|x| < 0.5 * epsilon_F/r_F$
$= \textbf{-0}$ if $x = \textbf{-0}$
$= +\infty$ if $x = +\infty$
$= -1$ if $x = -\infty$
$= \textbf{qNaN}$ if x is a quiet NaN
$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN

NOTES

2 **underflow** is explicitly avoided, when possible. ISO/IEC 10967-1:1994 requires that $fminN_F \leq epsilon_F$, but does *not* require that $fminN_F \leq epsilon_F/r_F$. A requirement that $expm1_F(x) = x$ if $x \in F$ and $|x| \leq fminN_F$, would thus be *requiring* results for some arguments of some (very rare) floating point type that are more than 0.5 ulp in error.

3 $expm1_F(1) = nearest_F(e - 1)$.

4 $expm1_F(x)$ will overflow approximately when $x > \ln(fmax_F)$.

5.3.3.3 Floating point power-of argument base operations

The $power^*_F$ approximation helper function:

$power^*_F : F \times F \to \mathcal{R}$

$power^*_F(x, y)$ 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 or equal to 0, and need not be defined when both of the arguments are zero.

Further requirements on the $power^*_F$ approximation helper function:

$power^*_F(1, y) = 1$ if $y \in F$
$power^*_F(x, 0) = 1$ if $x \in F$ and $x > 0$
$power^*_F(x, 1) = x$ if $x \in F$ and $x \geq 0$
$power^*_F(x, y) < fminD_F/2$ if $x \in F$ and $x > 0$ and $y \in F$ and $x^y < fminD_F/3$

The $power_F$ operation:

$power_F : F \times F \to F \cup \{\textbf{invalid}, \textbf{underflow}, \textbf{floating\_overflow}, \textbf{pole}\}$

$power_F(x, y) = trans\_result_F(power^*_F(x, y))$ if $x \in F$ and $x > 0$ and $y \in F$
$= power_F(0, y)$ if $x = \textbf{-0}$ and $y \in F \cup \{-\infty, \textbf{-0}, +\infty\}$
$= power_F(x, 0)$ if $y = \textbf{-0}$ and $x \in F \cup \{-\infty, +\infty\}$
$= 0$ if $x = 0$ and $y \in F$ and $y > 0$
$= \textbf{invalid}(1)$ if $x = 0$ and $y = 0$
$= \textbf{pole}(+\infty)$ if $x = 0$ and $y \in F$ and $y < 0$
$= 0$ if $x \in F$ and $0 \leq x < 1$ and $y = +\infty$

$= +\infty$	if $x \in F$ and $0 \le x < 1$ and $y = -\infty$
$=$ **invalid**(1)	if $x = 1$ and ($y = +\infty$ or $y = -\infty$)
$= +\infty$	if $x \in F$ and $x > 1$ and $y = +\infty$
$= 0$	if $x \in F$ and $x > 1$ and $y = -\infty$
$= +\infty$	if $x = +\infty$ and (($y \in F$ and $y > 0$) or $y = +\infty$)
$=$ **invalid**(1)	if $x = +\infty$ and $y = 0$
$= 0$	if $x = +\infty$ and (($y \in F$ and $y < 0$) or $y = -\infty$)
$=$ **invalid**$($**qNaN**$)$	if (($x \in F$ and $x < 0$) or $x = -\infty$) and $y \in F \cup \{+\infty, -\infty\}$
$=$ **qNaN**	if x is a quiet NaN and y is not a signalling NaN
$=$ **qNaN**	if y is a quiet NaN and x is not a signalling NaN
$=$ **invalid**$($**qNaN**$)$	if x is a signalling NaN or y is a signalling NaN

NOTE 1 – $power_F(x, y)$ 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 $x \in \{0, 1\}$.

The $power^*_{FI}$ approximation helper function:

$power^*_{FI} : F \times I \to \mathcal{R}$

$power^*_{FI}(x, y)$ returns a close approximation to x^y in $\mathcal{R}$, with maximum error $max\_error\_power_F$.

Further requirements on the $power^*_{FI}$ approximation helper function:

$power^*_{FI}(1, y) = 1$	if $y \in I$
$power^*_{FI}(x, 0) = 1$	if $x \in F$ and $x \neq 0$
$power^*_{FI}(x, 1) = x$	if $x \in F$
$power^*_{FI}(x, y) < fminD_F/2$	if $x \in F$ and $x > 0$ and $y \in I$ and $x^y < fminD_F/3$
$power^*_{FI}(x, y) = power^*_{FI}(-x, y)$	if $x \in F$ and $x < 0$ and $y \in I$ and $2 \mid y$
$power^*_{FI}(x, y) = -power^*_{FI}(-x, y)$	if $x \in F$ and $x < 0$ and $y \in I$ and not $2 \mid y$
$power^*_{FI}(x, y) = power^*_F(x, y)$	if $x \in F$ and $x > 0$ and $y \in I \cap F$

The $power_{FI}$ operation:

$power_{FI} : F \times I \to F \cup \{$**invalid**, **underflow**, **floating_overflow**, **pole**$\}$

$power_{FI}(x, y) = trans\_result_F(power^*_{FI}(x, y))$	if $x \in F$ and $x \neq 0$ and $y \in I$
$= 0$	if $x = 0$ and $y \in I$ and $y > 0$
$=$ **invalid**(1)	if $x = 0$ and $y = 0$
$=$ **pole**$(+\infty)$	if $x = 0$ and $y \in I$ and $y < 0$
$= 0$	if $x =$ **−0** and $y \in I$ and $y > 0$ and $2 \mid y$
$=$ **−0**	if $x =$ **−0** and $y \in I$ and $y > 0$ and not $2 \mid y$
$=$ **invalid**(1)	if $x =$ **−0** and $y = 0$
$=$ **pole**$(+\infty)$	if $x =$ **−0** and $y \in I$ and $y < 0$ and $2 \mid y$
$=$ **pole**$(-\infty)$	if $x =$ **−0** and $y \in I$ and $y < 0$ and not $2 \mid y$
$= +\infty$	if $x = +\infty$ and $y \in I$ and $y > 0$
$=$ **invalid**(1)	if $x = +\infty$ and $y = 0$
$= 0$	if $x = +\infty$ and $y \in I$ and $y < 0$
$= +\infty$	if $x = -\infty$ and $y \in I$ and $y > 0$ and $2 \mid y$
$= -\infty$	if $x = -\infty$ and $y \in I$ and $y > 0$ and not $2 \mid y$
$=$ **invalid**(1)	if $x = -\infty$ and $y = 0$
$= 0$	if $x = -\infty$ and $y \in I$ and $y < 0$ and $2 \mid y$
$=$ **−0**	if $x = -\infty$ and $y \in I$ and $y < 0$ and not $2 \mid y$
$=$ **qNaN**	if x is a quiet NaN
$=$ **invalid**$($**qNaN**$)$	if x is a signalling NaN

NOTES

2 $power_{FI}(x, y)$ 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 $x = 1$.

3 $power_I$ (in clause 5.1.4) does not allow negative *exponents* since the exact result then is not in $\mathcal{Z}$. $power_F$ 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.

5.3.3.4 Operation for power-of argument base, minus one

The $powerm1^*_F$ approximation helper function:

$powerm1^*_F : F \times F \to \mathcal{R}$

$powerm1^*_F(x, y)$ returns a close approximation to $x^y - 1$ in $\mathcal{R}$, with maximum error $max\_error\_power_F$. The $powerm1^*_F$ helper function need be defined only for first arguments that are greater than or equal to 0, and need not be defined when both of the arguments are zero.

NOTE 1 – There are two advantages with the $powerm1_F$ operation below: Firstly, $powerm1_F(b, x)$ are much more accurate than $sub_F(power_F(b, x), 1)$ when the exponent argument is close to zero. Secondly, the $powerm1_F$ operation does not **underflow** for "very" negative exponent arguments (when the base is greater than 1). Something which may be advantageous if **underflow** handling is slow, and high accuracy for "very" negative arguments is not needed.

Further requirements on the $powerm1^*_F$ approximation helper function:

$powerm1^*_F(0, y) = -1$	if $y \in F$ and $y > 0$
$powerm1^*_F(x, y) = -1$	if $x \in F$ and $x > 0$ and $y \in F$ and $powerm1^*_F(x, y) \neq x^y - 1$ and $x^y < epsilon_F/(3 * r_F)$
$powerm1^*_F(x, 1) = x - 1$	if $x \in F$ and $x \geq 0$
$powerm1^*_F(x, y) \leq power^*_F(x, y)$	if $x \in F$ and $x > 0$ and $y \in F$

NOTE 2 – $powerm1^*_F(x, y) \approx y * \ln(x)$ if $x \in F$ and $x > 0$ and $y \in F$ and $|y * \ln(x)| < epsilon_F/r_F$.

The $powerm1_F$ operation:

$powerm1_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{invalid}, \mathbf{underflow}, \mathbf{floating\_overflow}, \mathbf{pole}\}$

$powerm1_F(x, y)$

$= trans\_result_F(powerm1^*_F(x, y))$	if $x \in F$ and $x > 0$ and $y \in F$ and $y \neq 0$
$= powerm1_F(0, y)$	if $x = \mathbf{-0}$ and $y \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{-0}$	if $y = 0$ and $x \in F$ and $0 < x < 1$
$= 0$	if $y = 0$ and $x \in F$ and $1 \leq x$
$= 0$	if $y = 0$ and $x = +\infty$
$= 0$	if $y = \mathbf{-0}$ and $x \in F$ and $0 < x < 1$
$= \mathbf{-0}$	if $y = \mathbf{-0}$ and $x \in F$ and $1 \leq x$
$= \mathbf{-0}$	if $y = \mathbf{-0}$ and $x = +\infty$
$= -1$	if $x = 0$ and $y \in F$ and $y > 0$
$= \mathbf{invalid}(\mathbf{-0})$	if $x = 0$ and $y = 0$
$= \mathbf{invalid}(0)$	if $x = 0$ and $y = \mathbf{-0}$
$= \mathbf{pole}(+\infty)$	if $x = 0$ and $y \in F$ and $y < 0$
$= -1$	if $x \in F$ and $0 \leq x < 1$ and $y = +\infty$
$= +\infty$	if $x \in F$ and $0 \leq x < 1$ and $y = -\infty$

$= \mathbf{invalid}(0)$	if $x = 1$ and $(y = +\infty$ or $y = -\infty)$
$= +\infty$	if $x \in F$ and $x > 1$ and $y = +\infty$
$= -1$	if $x \in F$ and $x > 1$ and $y = -\infty$
$= +\infty$	if $x = +\infty$ and $((y \in F$ and $y > 0)$ or $y = +\infty)$
$= \mathbf{invalid}(0)$	if $x = +\infty$ and $y = 0$
$= -1$	if $x = +\infty$ and $((y \in F$ and $y < 0)$ or $y = -\infty)$
$= \mathbf{invalid}(\mathbf{qNaN})$	if $((x \in F$ and $x < 0)$ or $x = -\infty)$ and $y \in F \cup \{+\infty, \mathbf{-0}, -\infty\}$
$= \mathbf{qNaN}$	if x is a quiet NaN and y is not a signalling NaN
$= \mathbf{qNaN}$	if y is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$	if x is a signalling NaN or y is a signalling NaN

NOTE 3 – $powerm1_F(x, y)$ 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 $y < \log_x(fmax_F)$. It will not overflow when $x \in \{0, 1\}$.

5.3.3.5 Power-of 2 operation

The $exp2^*_F$ approximation helper function:

$exp2^*_F : F \to \mathcal{R}$

$exp2^*_F(x)$ 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:

$exp2^*_F(x) = 1$	if $x \in F$ and $exp2^*_F(x) \neq 2^x$ and $\log_2(1 - (epsilon_F/(2 * r_F))) < x$ and $x < \log_2(1 + (epsilon_F/2))$
$exp2^*_F(x) = 2^x$	if $x \in (F \cap Z)$ and $2^x \in F$
$exp2^*_F(x) < fminD_F/2$	if $x \in F$ and $x < \log_2(fminD_F) - 3$

The $exp2_F$ operation:

$exp2_F : F \to F \cup \{\mathbf{underflow}, \mathbf{floating\_overflow}\}$

$exp2_F(x) = trans\_result_F(exp2^*_F(x))$	if $x \in F$
$= 1$	if $x = \mathbf{-0}$
$= +\infty$	if $x = +\infty$
$= 0$	if $x = -\infty$
$= \mathbf{qNaN}$	if x is a quiet NaN
$= \mathbf{invalid}(\mathbf{qNaN})$	if x is a signalling NaN

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

5.3.3.6 Power-of 10 operation

The $exp10^*_F$ approximation helper function:

$exp10^*_F : F \to \mathcal{R}$

$exp10^*_F(x)$ 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:

$exp10^*_F(x) = 1$ if $x \in F$ and $exp10^*_F(x) \neq 10^x$ and $\log_{10}(1 - (epsilon_F/(2 * r_F))) < x$ and $x < \log_{10}(1 + (epsilon_F/2))$

$exp10^*_F(x) = 10^x$ if $x \in (F \cap Z)$ and $10^x \in F$

$exp10^*_F(x) < fminD_F/2$ if $x \in F$ and $x < \log_{10}(fminD_F) - 3$

The $exp10_F$ operation:

$exp10_F : F \to F \cup \{\textbf{underflow}, \textbf{floating\_overflow}\}$

$exp10_F(x) = trans\_result_F(exp10^*_F(x))$ if $x \in F$

$= 1$ if $x = \textbf{-0}$

$= +\infty$ if $x = +\infty$

$= 0$ if $x = -\infty$

$= \textbf{qNaN}$ if x is a quiet NaN

$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN

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

5.3.3.7 Natural logarithm-of operation

The ln^*_F approximation helper function:

$ln^*_F : \mathcal{R} \to \mathcal{R}$

$ln^*_F(x)$ returns a close approximation to $\ln(x)$ in $\mathcal{R}$, with maximum error $max\_error\_exp_F$.

Further requirements on the ln^*_F approximation helper function:

$ln^*_F(e) = 1$

The ln_F operation:

$ln_F : F \to F \cup \{\textbf{invalid}, \textbf{pole}\}$

$ln_F(x) = trans\_result_F(ln^*_F(x))$ if $x \in F$ and $x > 0$

$= \textbf{pole}(-\infty)$ if $x = 0$

$= \textbf{pole}(-\infty)$ if $x = \textbf{-0}$

$= +\infty$ if $x = +\infty$

$= \textbf{invalid}(\textbf{qNaN})$ if ($x \in F$ and $x < 0$) or $x = -\infty$

$= \textbf{qNaN}$ if x is a quiet NaN

$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN

5.3.3.8 Operation for natural logarithm-of one plus the argument

The $ln1p^*_F$ approximation helper function:

$ln1p^*_F : \mathcal{R} \to \mathcal{R}$

$ln1p^*_F(x)$ 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:

$ln1p^*_F(e - 1) = 1$

$ln1p^*_F(x) = x$ if $x \in F$ and $ln1p^*_F(x) \neq \ln(1 + x)$ and $-0.5 * epsilon_F/r_F < x \leq epsilon_F/r_F$

$ln1p^*_F(x) \geq ln^*_F(x)$ if $x \in F$ and $x > 0$

The $ln1p_F$ operation:

$ln1p_F : F \to F \cup \{\textbf{invalid}, \textbf{pole}, \textbf{underflow}\}$

$$
\begin{array}{lll}
ln1p_F(x) & = trans\_result_F(ln1p^*_F(x)) & \\
 & & \text{if } x \in F \text{ and } x > -1 \text{ and } |x| \geq 0.5 * epsilon_F/r_F \\
 & = x & \text{if } x \in F \text{ and } |x| < 0.5 * epsilon_F/r_F \\
 & = \mathbf{-0} & \text{if } x = \mathbf{-0} \\
 & = \mathbf{pole}(-\infty) & \text{if } x = -1 \\
 & = +\infty & \text{if } x = +\infty \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } (x \in F \text{ and } x < -1) \text{ or } x = -\infty \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTE – **underflow** is explicitly avoided, when possible. LIA-1 requires that $fminN_F \leq epsilon_F$, but does not require that $fminN_F \leq epsilon_F/r_F$. A requirement that $ln1p_F(x) = x$ if $x \in F$ and $|x| \leq fminN_F$ would thus be requireing results for some arguments of some (very rare) floating point type that are more than 0.5 ulp in error. For such arguments in such floating point types, underflow is still appropriate, and it is always appropriate to allow results that are at most 0.5 ulp in error.

5.3.3.9 Argument base logarithm-of operation

The $logbase^*_F$ approximation helper function:

$logbase^*_F : F \times F \to \mathcal{R}$

$logbase^*_F(x, y)$ returns a close approximation to $\log_x(y)$ in $\mathcal{R}$, with maximum error $max\_error\_power_F$.

Further requirements on the $logbase^*_F$ approximation helper function:

$logbase^*_F(x, x) = 1$ if $x \in F$ and $x > 0$ and $x \neq 1$

The $logbase_F$ operation:

$logbase_F : F \times F \to F \cup \{\mathbf{invalid}, \mathbf{pole}\}$

$$
\begin{array}{lll}
logbase_F(x, y) & = trans\_result_F(logbase^*_F(x, y)) & \\
 & & \text{if } x \in F \text{ and } x > 0 \text{ and } x \neq 1 \text{ and} \\
 & & \quad y \in F \text{ and } y > 0 \\
 & = logbase_F(0, y) & \text{if } x = \mathbf{-0} \text{ and } y \in F \cup \{-\infty, \mathbf{-0}, +\infty\} \\
 & = logbase_F(x, 0) & \text{if } y = \mathbf{-0} \text{ and } x \in F \cup \{-\infty, +\infty\} \\
 & = \mathbf{pole}(+\infty) & \text{if } x \in F \text{ and } 0 < x < 1 \text{ and } y = 0 \\
 & = \mathbf{pole}(-\infty) & \text{if } x \in F \text{ and } 1 < x \text{ and } y = 0 \\
 & = -\infty & \text{if } x \in F \text{ and } 0 < x < 1 \text{ and } y = +\infty \\
 & = +\infty & \text{if } x \in F \text{ and } 1 < x \text{ and } y = +\infty \\
 & = \mathbf{invalid}(1) & \text{if } x = +\infty \text{ and } y = +\infty \\
 & = 0 & \text{if } x = +\infty \text{ and } y \in F \text{ and } y \geq 1 \\
 & = \mathbf{-0} & \text{if } x = +\infty \text{ and } y \in F \text{ and } 0 < y < 1 \\
 & = \mathbf{invalid}(-1) & \text{if } x = +\infty \text{ and } y = 0 \\
 & = +\infty & \text{if } x = 1 \text{ and } y = +\infty \\
 & = \mathbf{pole}(+\infty) & \text{if } x = 1 \text{ and } y \in F \text{ and } y > 1 \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x = 1 \text{ and } y = 1 \\
 & = \mathbf{pole}(-\infty) & \text{if } x = 1 \text{ and } y \in F \text{ and } 0 \leq y < 1 \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } ((x \in F \text{ and } x \leq 0) \text{ or } x = -\infty) \text{ and} \\
 & & \quad y \in F \cup \{+\infty, -\infty\} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } ((y \in F \text{ and } y < 0) \text{ or } y = -\infty) \text{ and} \\
 & & \quad x \in F \cup \{+\infty, -\infty\} \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN and } y \text{ is not a signalling NaN} \\
 & = \mathbf{qNaN} & \text{if } y \text{ is a quiet NaN and } x \text{ is not a signalling NaN}
\end{array}
$$

$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN

5.3.3.10 Operation for argument base logarithm-of one plus second argument

The $logbase1p^*_F$ approximation helper function:

$logbase1p^*_F : F \times F \to \mathcal{R}$

$logbase1p^*_F(x, y)$ returns a close approximation to $\log_x(1 + y)$ in $\mathcal{R}$, with maximum error $max\_error\_power_F$.

Further requirements on $logbase1p^*_F$ approximating helper function:

$logbase1p^*_F(x, x - 1) = 1$ if $x, x - 1 \in F$ and $x > 0$ and $x \neq 1$
$logbase1p^*_F(x, y) \leq logbase^*_F(x, y)$ if $x \in F$ and $0 < x < 1$ and $y \in F$ and $y > 0$
$logbase1p^*_F(x, y) \geq logbase^*_F(x, y)$ if $x \in F$ and $1 < x$ and $y \in F$ and $y > 0$

NOTE – $logbase1p^*_F(x, y) \approx y/\ln(x)$ if $x \in F$ and $x > 0$ and $x \neq 1$ and $y \in F$ and $|y/\ln(x)| < epsilon_F/r_F$.

The $logbase1p_F$ operation:

$logbase1p_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{invalid}, \mathbf{underflow}, \mathbf{pole}\}$

$logbase1p_F(x, y)$

$= trans\_result_F(logbase1p^*_F(x, y))$ if $x \in F$ and $x > 0$ and $x \neq 1$ and $y \in F$ and $y > -1$ and $y \neq 0$
$= logbase1p_F(0, y)$ if $x = \mathbf{-0}$ and $y \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{-0}$ if $y = 0$ and $x \in F$ and $0 < x < 1$
$= 0$ if $y = 0$ and $x \in F$ and $1 < x$
$= 0$ if $y = 0$ and $x = +\infty$
$= 0$ if $y = \mathbf{-0}$ and $x \in F$ and $0 < x < 1$
$= \mathbf{-0}$ if $y = \mathbf{-0}$ and $x \in F$ and $1 < x$
$= \mathbf{-0}$ if $y = \mathbf{-0}$ and $x = +\infty$
$= \mathbf{invalid}(\mathbf{qNaN})$ if ($y = \mathbf{-0}$ or $y = 0$) and $x = 1$
$= \mathbf{pole}(+\infty)$ if $x \in F$ and $0 < x < 1$ and $y = -1$
$= \mathbf{pole}(-\infty)$ if $x \in F$ and $1 < x$ and $y = -1$
$= -\infty$ if $x \in F$ and $0 < x < 1$ and $y = +\infty$
$= +\infty$ if $x \in F$ and $1 < x$ and $y = +\infty$
$= \mathbf{invalid}(1)$ if $x = +\infty$ and $y = +\infty$
$= 0$ if $x = +\infty$ and $y \in F$ and $y \geq 0$
$= \mathbf{-0}$ if $x = +\infty$ and $y \in F$ and $-1 < y < 0$
$= \mathbf{invalid}(-1)$ if $x = +\infty$ and $y = -1$
$= +\infty$ if $x = 1$ and $y = +\infty$
$= \mathbf{pole}(+\infty)$ if $x = 1$ and $y \in F$ and $y > 0$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $x = 1$ and $y = 0$
$= \mathbf{pole}(-\infty)$ if $x = 1$ and $y \in F$ and $-1 \leq y < 0$
$= \mathbf{invalid}(\mathbf{qNaN})$ if (($x \in F$ and $x(0)$ or $x = -\infty$) and $y \in F \cup \{+\infty, \mathbf{-0}, -\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if (($y \in F$ and $y < -1$) or $y = -\infty$) and $x \in F \cup \{+\infty, -\infty\}$
$= \mathbf{qNaN}$ if x is a quiet NaN and y is not a signalling NaN
$= \mathbf{qNaN}$ if y is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or y is a signalling NaN

5.3.3.11 2-logarithm-of operation

The $log2^*_F$ approximation helper function:

$log2^*_F : F \to \mathcal{R}$

$log2^*_F(x)$ returns a close approximation to $\log_2(x)$ in $\mathcal{R}$, with maximum error $max\_error\_exp_F$.

Further requirements on the $log2^*_F$ approximation helper function:

$log2^*_F(x) = \log_2(x)$ if $x \in F$ and $\log_2(x) \in \mathcal{Z}$

The $log2_F$ operation:

$log2_F : F \to F \cup \{\textbf{invalid}, \textbf{pole}\}$

$$
\begin{array}{lll}
log2_F(x) & = trans\_result_F(log2^*_F(x)) & \\
 & & \text{if } x \in F \text{ and } x > 0 \\
 & = \textbf{pole}(-\infty) & \text{if } x = 0 \\
 & = \textbf{pole}(-\infty) & \text{if } x = \textbf{-0} \\
 & = +\infty & \text{if } x = +\infty \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } (x \in F \text{ and } x < 0) \text{ or } x = -\infty \\
 & = \textbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

5.3.3.12 10-logarithm-of operation

The $log10^*_F$ approximation helper function:

$log10^*_F : F \to \mathcal{R}$

$log10^*_F(x)$ returns a close approximation to $\log_{10}(x)$ in $\mathcal{R}$, with maximum error $max\_error\_exp_F$.

Further requirements on the $log10^*_F$ approximation helper function:

$log10^*_F(x) = \log_{10}(x)$ if $x \in F$ and $\log_{10}(x) \in \mathcal{Z}$

The $log10_F$ operation:

$log10_F : F \to F \cup \{\textbf{invalid}, \textbf{pole}\}$

$$
\begin{array}{lll}
log10_F(x) & = trans\_result_F(log10^*_F(x)) & \\
 & & \text{if } x \in F \text{ and } x > 0 \\
 & = \textbf{pole}(-\infty) & \text{if } x = 0 \\
 & = \textbf{pole}(-\infty) & \text{if } x = \textbf{-0} \\
 & = +\infty & \text{if } x = +\infty \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } (x \in F \text{ and } x < 0) \text{ or } x = -\infty \\
 & = \textbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

5.3.4 Operations for hyperbolics and inverse hyperbolics

There are two maximum error parameters for operations corresponding to the hyperbolic and inverse hyperbolic functions:

$max\_error\_sinh_F \in F$
$max\_error\_tanh_F \in F$

The $max\_error\_sinh_F$ parameter is required to be in the interval $[0.5, 2 * rnd\_error_F]$. The $max\_error\_tanh_F$ parameter is required to be in the interval $[max\_error\_sinh_F, 2 * rnd\_error_F]$.

5.3.4.1 Sinus hyperbolicus operation

The $sinh_F^*$ approximation helper function:

$sinh_F^* : F \to \mathcal{R}$

$sinh_F^*(x)$ returns a close approximation to $\sinh(x)$ in $\mathcal{R}$, with maximum error $max\_error\_sinh_F$.

Further requirements on the $sinh_F^*$ approximation helper function:

$sinh_F^*(x) = x$	if $x \in F$ and $sinh_F^*(x) \neq \sinh(x)$ and $\lvert x\rvert < \sqrt{2 * epsilon_F / r_F}$
$sinh_F^*(-x) = -sinh_F^*(x)$	if $x \in F$
$sinh_F^*(x) \leq cosh_F^*(x)$	if $x \in F$

The $sinh_F$ operation:

$sinh_F : F \to F \cup \{\textbf{floating\_overflow}\}$

$sinh_F(x)$	$= trans\_result_F(sinh_F^*(x))$	if $x \in F$ and $\lvert x\rvert > fminN_F$
	$= x$	if $x \in F$ and $\lvert x\rvert \leq fminN_F$
	$= \textbf{-0}$	if $x = \textbf{-0}$
	$= -\infty$	if $x = -\infty$
	$= +\infty$	if $x = +\infty$
	$= \textbf{qNaN}$	if x is a quiet NaN
	$= \textbf{invalid}(\textbf{qNaN})$	if x is a signalling NaN

NOTES

1 **underflow** is explicitly avoided.

2 $sinh_F(x)$ will overflow approximately when $\lvert x\rvert > \ln(2 * fmax_F)$.

5.3.4.2 Cosinus hyperbolicus operation

The $cosh_F^*$ approximation helper function:

$cosh_F^* : F \to \mathcal{R}$

$cosh_F^*(x)$ returns a close approximation to $\cosh(x)$ in $\mathcal{R}$, with maximum error $max\_error\_sinh_F$.

Further requirements on the $cosh_F^*$ approximation helper function:

$cosh_F^*(x) = 1$	if $x \in F$ and $cosh_F^*(x) \neq \cosh(x)$ and $\lvert x\rvert < \sqrt{epsilon_F}$
$cosh_F^*(-x) = cosh_F^*(x)$	if $x \in F$
$cosh_F^*(x) \geq sinh_F^*(x)$	if $x \in F$

The $cosh_F$ operation:

$cosh_F : F \to F \cup \{\textbf{floating\_overflow}\}$

$cosh_F(x)$	$= trans\_result_F(cosh_F^*(x))$	if $x \in F$
	$= 1$	if $x = \textbf{-0}$
	$= +\infty$	if $x = -\infty$
	$= +\infty$	if $x = +\infty$
	$= \textbf{qNaN}$	if x is a quiet NaN
	$= \textbf{invalid}(\textbf{qNaN})$	if x is a signalling NaN

NOTE – $cosh_F(x)$ overflows approximately when $\lvert x\rvert > \ln(2 * fmax_F)$.

5.3.4.3 Tangentus hyperbolicus operation

The $tanh_F^*$ approximation helper function:

$tanh_F^* : F \to \mathcal{R}$

$tanh_F^*(x)$ returns a close approximation to $\tanh(x)$ in $\mathcal{R}$, with maximum error $max\_error\_tanh_F$.

Further requirements on the $tanh_F^*$ approximation helper function:

$tanh_F^*(x) = x$	if $x \in F$ and $tanh_F^*(x) \neq \tanh(x)$ and $\lvert x \rvert \leq \sqrt{1.5 * epsilon_F / r_F}$
$tanh_F^*(x) = 1$	if $x \in F$ and $tanh_F^*(x) \neq \tanh(x)$ and $x > arctanh(1 - (epsilon_F/(3 * r_F)))$
$tanh_F^*(-x) = -tanh_F^*(x)$	if $x \in F$

The $tanh_F$ operation:

$tanh_F : F \to F$

$tanh_F(x)$	$= trans\_result_F(tanh_F^*(x))$	if $x \in F$ and $\lvert x \rvert > fminN_F$
	$= x$	if $x \in F$ and $\lvert x \rvert \leq fminN_F$
	$= \mathbf{-0}$	if $x = \mathbf{-0}$
	$= -1$	if $x = -\infty$
	$= 1$	if $x = +\infty$
	$= \mathbf{qNaN}$	if x is a quiet NaN
	$= \mathbf{invalid(qNaN)}$	if x is a signalling NaN

NOTE – **underflow** is explicitly avoided.

5.3.4.4 Cotangentus hyperbolicus operation

The $coth_F^*$ approximation helper function:

$coth_F^* : F \to \mathcal{R}$

$coth_F^*(x)$ returns a close approximation to $\coth(x)$ in $\mathcal{R}$, with maximum error $max\_error\_tanh_F$.

Further requirements on the $coth_F^*$ approximation helper function:

$coth_F^*(x) = 1$	if $x \in F$ and $coth_F^*(x) \neq \coth(x)$ and $x > arccoth(1 + (epsilon_F/4))$
$coth_F^*(-x) = -coth_F^*(x)$	if $x \in F$

The $coth_F$ operation:

$coth_F : F \to F \cup \{\mathbf{pole}, \mathbf{floating\_overflow}\}$

$coth_F(x)$	$= trans\_result_F(coth_F^*(x))$	if $x \in F$ and $x \neq 0$
	$= \mathbf{pole}(+\infty)$	if $x = 0$
	$= \mathbf{pole}(-\infty)$	if $x = \mathbf{-0}$
	$= -1$	if $x = -\infty$
	$= 1$	if $x = +\infty$
	$= \mathbf{qNaN}$	if x is a quiet NaN
	$= \mathbf{invalid(qNaN)}$	if x is a signalling NaN

NOTE – $coth_F(x)$ overflow approximately when $\lvert 1/x \rvert > fmax_F$.

5.3.4.5 Secantus hyperbolicus operation

The $sech^*_F$ approximation helper function:

$sech^*_F : F \to \mathcal{R}$

$sech^*_F(x)$ returns a close approximation to $sech(x)$ in $\mathcal{R}$, with maximum error $max\_error\_tanh_F$.

Further requirements on the $sech^*_F$ approximation helper function:

$sech^*_F(x) = 1$ if $x \in F$ and $sech^*_F(x) \neq sech(x)$ and $|x| < \sqrt{epsilon_F/r_F}$
$sech^*_F(-x) = sech^*_F(x)$ if $x \in F$
$sech^*_F(x) \leq csch^*_F(x)$ if $x \in F$ and $x > 0$
$sech^*_F(x) < fminD_F/2$ if $x \in F$ and $x > 2 - \ln(fminD_F/4)$

The $sech_F$ operation:

$sech_F : F \to F \cup \{\textbf{underflow}\}$

$sech_F(x) = trans\_result_F(sech^*_F(x))$ if $x \in F$
$= 1$ if $x = \textbf{-0}$
$= 0$ if $x = -\infty$
$= 0$ if $x = +\infty$
$= \textbf{qNaN}$ if x is a quiet NaN
$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN

5.3.4.6 Cosecantus hyperbolicus operation

The $csch^*_F$ approximation helper function:

$csch^*_F : F \to \mathcal{R}$

$csch^*_F(x)$ returns a close approximation to $csch(x)$ in $\mathcal{R}$, with maximum error $max\_error\_tanh_F$.

Further requirements on the $csch^*_F$ approximation helper function:

$csch^*_F(-x) = -csch^*_F(x)$ if $x \in F$
$csch^*_F(x) \geq sech^*_F(x)$ if $x \in F$ and $x > 0$
$csch^*_F(x) < fminD_F/2$ if $x \in F$ and $x > 2 - \ln(fminD_F/4)$

The $csch_F$ operation:

$csch_F : F \to F \cup \{\textbf{underflow}, \textbf{floating\_overflow}, \textbf{pole}\}$

$csch_F(x) = trans\_result_F(csch^*_F(x))$ if $x \in F$ and $x \neq 0$
$= \textbf{pole}(+\infty)$ if $x = 0$
$= \textbf{pole}(-\infty)$ if $x = \textbf{-0}$
$= \textbf{-0}$ if $x = -\infty$
$= 0$ if $x = +\infty$
$= \textbf{qNaN}$ if x is a quiet NaN
$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN

NOTE – $csch_F(x)$ overflows approximately when $|1/x| > fmax_F$.

5.3.4.7 Arcus sinus hyperbolicus operation

The $arcsinh_F^*$ approximation helper function:

$arcsinh_F^* : F \to \mathcal{R}$

$arcsinh_F^*(x)$ returns a close approximation to $arcsinh(x)$ in $\mathcal{R}$, with maximum error $max\_error\_sinh_F$.

Further requirements on the $arcsinh_F^*$ approximation helper function:

$arcsinh_F^*(x) = x$ if $x \in F$ and $arcsinh_F^*(x) \neq arcsinh(x)$ and $|x| \leq \sqrt{3 * epsilon_F / r_F}$

$arcsinh_F^*(-x) = -arcsinh_F^*(x)$ if $x \in F$

The $arcsinh_F$ operation:

$arcsinh_F : F \to F$

$$
\begin{array}{lll}
arcsinh_F(x) & = trans\_result_F(arcsinh_F^*(x)) & \\
 & & \text{if } x \in F \text{ and } |x| > fminN_F \\
 & = x & \text{if } x \in F \text{ and } |x| \leq fminN_F \\
 & = \mathbf{-0} & \text{if } x = \mathbf{-0} \\
 & = -\infty & \text{if } x = -\infty \\
 & = +\infty & \text{if } x = +\infty \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTE – **underflow** is explicitly avoided.

5.3.4.8 Arcus cosinus hyperbolicus operation

The $arccosh_F^*$ approximation helper function:

$arccosh_F^* : F \to \mathcal{R}$

$arccosh_F^*(x)$ returns a close approximation to $arccosh(x)$ in $\mathcal{R}$, with maximum error $max\_error\_sinh_F$.

The $arccosh_F$ operation:

$arccosh_F : F \to F \cup \{\mathbf{invalid}\}$

$$
\begin{array}{lll}
arccosh_F(x) & = trans\_result_F(arccosh_F^*(x)) & \\
 & & \text{if } x \in F \text{ and } x \geq 1 \\
 & = +\infty & \text{if } x = +\infty \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \in F \text{ and } x < 1 \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x = \mathbf{-0} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x = -\infty \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

5.3.4.9 Arcus tangentus hyperbolicus operation

The $arctanh_F^*$ approximation helper function:

$arctanh_F^* : F \to \mathcal{R}$

$arctanh_F^*(x)$ returns a close approximation to $arctanh(x)$ in $\mathcal{R}$, with maximum error $max\_error\_tanh_F$.

Further requirements on the $arctanh_F^*$ approximation helper function:

$arctanh_F^*(x) = x$ if $x \in F$ and $arctanh_F^*(x) \neq arctanh(x)$ and $|x| < \sqrt{1 * epsilon_F / r_F}$

$arctanh_F^*(-x) = -arctanh_F^*(x)$ if $x \in F$

The $arctanh_F$ operation:

$arctanh_F : F \to F \cup \{\mathbf{invalid}, \mathbf{pole}\}$

$$
\begin{array}{lll}
arctanh_F(x) & = trans\_result_F(arctanh_F^*(x)) & \\
 & & \text{if } x \in F \text{ and } fminN_F < |x| < 1 \\
 & = x & \text{if } x \in F \text{ and } |x| \leq fminN_F \\
 & = \mathbf{pole}(+\infty) & \text{if } x = 1 \\
 & = \mathbf{pole}(-\infty) & \text{if } x = -1 \\
 & = \mathbf{-0} & \text{if } x = \mathbf{-0} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \in F \text{ and } |x| > 1 \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x = -\infty \text{ or } x = +\infty \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTE – **underflow** is explicitly avoided.

5.3.4.10 Arcus cotangentus hyperbolicus operation

The $arccoth_F^*$ approximation helper function:

$arccoth_F^* : F \to \mathcal{R}$

$arccoth_F^*(x)$ returns a close approximation to $arccoth(x)$ in $\mathcal{R}$, with maximum error $max\_error\_tanh_F$.

Further requirements on the $arccoth_F^*$ approximation helper function:

$arccoth_F^*(-x) = -arccoth_F^*(x)$ if $x \in F$

The $arccoth_F$ operation:

$arccoth_F : F \to F \cup \{\mathbf{invalid}, \mathbf{underflow}, \mathbf{pole}\}$

$$
\begin{array}{lll}
arccoth_F(x) & = trans\_result_F(arccoth_F^*(x)) & \\
 & & \text{if } x \in F \text{ and } |x| > 1 \\
 & = \mathbf{pole}(+\infty) & \text{if } x = 1 \\
 & = \mathbf{pole}(-\infty) & \text{if } x = -1 \\
 & = \mathbf{-0} & \text{if } x = -\infty \\
 & = 0 & \text{if } x = +\infty \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \in F \text{ and } -1 < x < 1 \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x = \mathbf{-0} \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTE – There is no **underflow** for this operation for most kinds of floating point types, e.g. IEC 559 ones.

5.3.4.11 Arcus secantus hyperbolicus operation

The $arcsech_F^*$ approximation helper function:

$arcsech_F^* : F \to \mathcal{R}$

$arcsech^*_F(x)$ returns a close approximation to with maximum error $max\_error\_tanh_F$.

The $arcsech_F$ operation:

$arcsech_F : F \to F \cup \{\mathbf{invalid}, \mathbf{pole}\}$

$$
\begin{array}{lll}
arcsech_F(x) & = trans\_result_F(arcsech^*_F(x)) & \\
 & & \text{if } x \in F \text{ and } 0 < x \leq 1 \\
 & = \mathbf{pole}(+\infty) & \text{if } x = 0 \\
 & = \mathbf{pole}(+\infty) & \text{if } x = \mathbf{-0} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \in F \text{ and } (x < 0 \text{ or } x > 1) \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x = -\infty \text{ or } x = +\infty \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

5.3.4.12 Arcus cosecantus hyperbolicus operation

The $arccsch^*_F$ approximation helper function:

$arccsch^*_F : F \to \mathcal{R}$

$arccsch^*_F(x)$ returns a close approximation to $arccsch(x)$ in $\mathcal{R}$, with maximum error $max\_error\_tanh_F$.

Further requirements on the $arccsch^*_F$ approximation helper function:

$$
\begin{array}{ll}
arccsch^*_F(1) = arcsinh^*_F(1) & \\
arccsch^*_F(-x) = -arccsch^*_F(x) & \text{if } x \in F
\end{array}
$$

The $arccsch_F$ operation:

$arccsch_F : F \to F \cup \{\mathbf{underflow}, \mathbf{pole}\}$

$$
\begin{array}{lll}
arccsch_F(x) & = trans\_result_F(arccsch^*_F(x)) & \\
 & & \text{if } x \in F \text{ and } x \leq 0 \\
 & = \mathbf{pole}(+\infty) & \text{if } x = 0 \\
 & = \mathbf{pole}(-\infty) & \text{if } x = \mathbf{-0} \\
 & = \mathbf{-0} & \text{if } x = -\infty \\
 & = 0 & \text{if } x = +\infty \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTE – There is no **underflow** for this operation for most kinds of floating point types, e.g. IEC 559 ones.

5.3.5 Introduction to operations for trigonometrics

The mathematical trigonometric functions are perfectly cyclic. Their numerical counterparts are not that perfect, for two reasons.

Firstly, the radian normalisation cannot be exact, even though it can be made very good given very many digits for the approximation(s) of π used in the angle normalisation, returning an offset from the nearest axis, and including guard digits. The unit argument normalisation, however, can be made exact regardless of the (non-zero and, in case $denorm_F = \mathbf{false}$, not too small) unit and the original angle, returning only a plain angle in F. ISO/IEC 10967-2 requires unit argument angle normalisation to be exact.

Secondly, the length of one revolution is of course constant, but the density of floating point values gets sparser (in absolute spacing rather than relative) the larger the magnitude of the values are. This means that the number of floating point values gets sparser per revolution the larger

the magnitude of the angle value. For this reason the notification **angle_too_big** is introduced. This notification is given when the magnitude of the angle value is "too big". Exactly when the representable angle values get too sparse may depend upon the application at hand, and it may be possible for the programmer to tighten the big-angle parameters below.

The continuation value upon an **angle_too_big** notification shall be **qNaN**.

Three different operations for each the 'conventional textbook' trigonometric functions are specified. One version for radians, one version where the angular unit is given as a parameter, and one where the angular unit is degrees.

5.3.6 Operations for radian trigonometrics and inverse radian trigonometrics

There shall be one radian big-angle parameter:

$big\_angle\_r_F \in F$

It shall have the following default value:

$big\_angle\_r_F = r_F^{\lceil p_F/2 \rceil}$

NOTE – The user may be allowed to narrow this value, but should not be allowed to widen it beyond the value given here.

The radian trigonometric approximation helper functions (including those for normalisation and conversion from radians) are required to have the same zero points as the approximated mathematical function only if the absolute value of the argument is less than or equal to $big\_angle\_r_F$. Likewise, the radian trigonometric approximation helper functions are required to have the same sign as the approximated mathematical function only if the absolute value of the argument is less than or equal to $big\_angle\_r_F$.

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

$max\_error\_sin_F \in F$
$max\_error\_tan_F \in F$

The $max\_error\_sin_F$ parameter shall be in the interval $[0.5, 1.5 * rnd\_error_F]$.

The $max\_error\_tan_F$ parameter shall be in the interval $[max\_error\_sin_F, 2 * rnd\_error_F]$.

5.3.6.1 Radian angle normalisation operations

The rad^*_F and $axis\_rad^*_F$ approximation helper functions have the signatures:

$rad^*_F : \mathcal{R} \to \mathcal{R}$
$axis\_rad^*_F : \mathcal{R} \to \{(1,0),(0,1),(-1,0),(0,-1)\} \times \mathcal{R}$

$rad^*_F(x)$ returns a close approximation to $rad(x)$ in $\mathcal{R}$, if $|x| \leq big\_angle\_r_F$, with maximum error $max\_error\_sin_F$.

$axis\_rad^*_F(x)$ returns a close approximation to $axis\_rad(x)$, if $x \leq big\_angle\_r_F$. The approximation consists of that the second part of the result (the offset from the indicated axis) is approximate.

Further requirements on the rad^*_F and $axis\_rad^*_F$ approximation helper functions:

$rad^*_F(x) = x$ if $|x| < \pi$
$snd(axis\_rad^*_F(x)) = rad^*_F(x)$ if $fst(axis\_rad^*_F(x)) = (1,0)$

The rad_F operation:

$rad_F : F \to F \cup \{\textbf{underflow}, \textbf{angle\_too\_big}\}$

$$
\begin{array}{lll}
rad_F(x) & = trans\_result_F(rad^*_F(x)) & \text{if } x \in F \text{ and } |x| > fminN_F \text{ and } |x| \le big\_angle\_r_F \\
& = x & \text{if } (x \in F \text{ and } |x| \le fminN_F) \text{ or } x = \mathbf{-0} \\
& = \mathbf{angle\_too\_big(qNaN)} & \text{if } x \in F \text{ and } |x| > big\_angle\_r_F \\
& = \mathbf{invalid(qNaN)} & \text{if } x \in \{-\infty, +\infty\} \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid(qNaN)} & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

The $axis\_rad_F$ operation:

$$axis\_rad_F : F \to ((F \times F) \times F) \cup \{\mathbf{angle\_too\_big}\}$$

$$
\begin{array}{lll}
axis\_rad_F(x) & = (fst(axis\_rad^*_F(x)), trans\_result_F(snd(axis\_rad^*_F(x)))) & \\
& & \text{if } x \in F \text{ and } |x| > fminN_F \text{ and } |x| \le big\_angle\_r_F \\
& = ((1,0),x) & \text{if } (x \in F \text{ and } |x| \le fminN_F) \text{ or } x = \mathbf{-0} \\
& = \mathbf{angle\_too\_big}((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN}) & \\
& & \text{if } x \in F \text{ and } |x| > big\_angle\_r_F \\
& = \mathbf{invalid}((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN}) & \\
& & \text{if } x \in \{-\infty, +\infty\} \\
& = ((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN}) & \\
& & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid}((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN}) & \\
& & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTE – 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. Hence these operations are not used in the specifications for the radian trigonometric operations.

5.3.6.2 Radian sinus operation

The sin^*_F approximation helper function:

$$sin^*_F : \mathcal{R} \to \mathcal{R}$$

$sin^*_F(x)$ returns a close approximation to $\sin(x)$ in $\mathcal{R}$ if $|x| \le big\_angle\_r_F$, with maximum error $max\_error\_sin_F$.

Further requirements on the sin^*_F approximation helper function:

$$
\begin{array}{ll}
sin^*_F(n * 2 * \pi + \pi/6) = 1/2 & \text{if } n \in \mathcal{Z} \text{ and } |n * 2 * \pi + \pi/6| \le big\_angle\_r_F \\
sin^*_F(n * 2 * \pi + \pi/4) = 1 & \text{if } n \in \mathcal{Z} \text{ and } |n * 2 * \pi + \pi/4| \le big\_angle\_r_F \\
sin^*_F(n * 2 * \pi + 5 * \pi/6) = 1/2 & \text{if } n \in \mathcal{Z} \text{ and } |n * 2 * \pi + 5 * \pi/6| \le big\_angle\_r_F \\
sin^*_F(x) = x & \text{if } sin^*_F(x) \ne \sin(x) \text{ and } |x| \le \sqrt{3 * epsilon_F / r_F} \\
sin^*_F(-x) = -sin^*_F(x) &
\end{array}
$$

The sin_F operation:

$$sin_F : F \to F \cup \{\mathbf{underflow}, \mathbf{angle\_too\_big}\}$$

$$
\begin{array}{lll}
sin_F(x) & = trans\_result_F(sin^*_F(x)) & \text{if } x \in F \text{ and } fminN_F < |x| \text{ and } |x| \le big\_angle\_r_F \\
& = x & \text{if } x \in F \text{ and } |x| \le fminN_F \\
& = \mathbf{-0} & \text{if } x = \mathbf{-0} \\
& = \mathbf{angle\_too\_big(qNaN)} & \text{if } |x| > big\_angle\_r_F \\
& = \mathbf{invalid(qNaN)} & \text{if } x \in \{-\infty, +\infty\} \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid(qNaN)} & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

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

5.3.6.3 Radian cosinus operation

The cos_F^* approximation helper function:

$cos_F^* : \mathcal{R} \to \mathcal{R}$

$cos_F^*(x)$ returns a close approximation to $\cos(x)$ in $\mathcal{R}$ if $|x| \leq big\_angle\_r_F$, with maximum error $max\_error\_sin_F$.

Further requirements on the cos_F^* approximation helper function:

$cos_F^*(n * 2 * \pi) = 1$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi| \leq big\_angle\_r_F$
$cos_F^*(n * 2 * \pi + \pi/3) = 1/2$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi/3| \leq big\_angle\_r_F$
$cos_F^*(n * 2 * \pi + 2 * \pi/3) = -1/2$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + 2 * \pi/3| \leq big\_angle\_r_F$
$cos_F^*(n * 2 * \pi + \pi) = -1$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi| \leq big\_angle\_r_F$
$cos_F^*(x) = 1$ if $cos_F^*(x) \neq \cos(x)$ and $|x| < \sqrt{epsilon_F / r_F}$
$cos_F^*(-x) = cos_F^*(x)$

The cos_F operation:

$cos_F : F \to F \cup \{\textbf{underflow}, \textbf{angle\_too\_big}\}$

$cos_F(x) = trans\_result_F(cos_F^*(x))$ if $x \in F$ and $|x| \leq big\_angle\_r_F$
$= 1$ if $x = \textbf{-0}$
$= \textbf{angle\_too\_big}(\textbf{qNaN})$ if $|x| > big\_angle\_r_F$
$= \textbf{invalid}(\textbf{qNaN})$ if $x \in \{-\infty, +\infty\}$
$= \textbf{qNaN}$ if x is a quiet NaN
$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN

5.3.6.4 Radian cosinus with sinus operation

$cossin_F : F \to (F \times F) \cup \{\textbf{underflow}, \textbf{angle\_too\_big}\}$
$cossin_F(x) = (cos_F(x), sin_F(x))$

5.3.6.5 Radian tangentus operation

The tan_F^* approximation helper function:

$tan_F^* : \mathcal{R} \to \mathcal{R}$

$tan_F^*(x)$ returns a close approximation to $\tan(x)$ in $\mathcal{R}$ if $|x| \leq big\_angle\_r_F$, with maximum error $max\_error\_tan_F$.

Further requirements on the tan_F^* approximation helper function:

$tan_F^*(n * 2 * \pi + \pi/4) = 1$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi/4| \leq big\_angle\_r_F$
$tan_F^*(n * 2 * \pi + 3 * \pi/4) = -1$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + 3 * \pi/4| \leq big\_angle\_r_F$
$tan_F^*(x) = x$ if $tan_F^*(x) \neq \tan(x)$ and $|x| < \sqrt{epsilon_F / r_F}$
$tan_F^*(-x) = -tan_F^*(x)$

The tan_F operation:

$tan_F : F \to F \cup \{\textbf{underflow}, \textbf{floating\_overflow}, \textbf{angle\_too\_big}\}$

$$
\begin{array}{lll}
tan_F(x) & = trans\_result_F(tan^*_F(x)) & \text{if } x \in F \text{ and } fminN_F < |x| \text{ and } |x| \le big\_angle\_r_F \\
& = x & \text{if } x \in F \text{ and } |x| \le fminN_F \\
& = \mathbf{-0} & \text{if } x = \mathbf{-0} \\
& = \mathbf{angle\_too\_big(qNaN)} & \text{if } |x| > big\_angle\_r_F \\
& = \mathbf{invalid(qNaN)} & \text{if } x \in \{-\infty, +\infty\} \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid(qNaN)} & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

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

5.3.6.6 Radian cotangentus operation

The cot^*_F approximation helper function:

$cot^*_F : \mathcal{R} \to \mathcal{R}$

$cot^*_F(x)$ returns a close approximation to $\cot(x)$ in $\mathcal{R}$ if $|x| \le big\_angle\_r_F$, with maximum error $max\_error\_tan_F$.

Further requirements on the cot^*_F approximation helper function:

$$
\begin{array}{ll}
cot^*_F(n * 2 * \pi + \pi/4) = 1 & \text{if } n \in \mathcal{Z} \text{ and } |n * 2 * \pi + \pi/4| \le big\_angle\_r_F \\
cot^*_F(n * 2 * \pi + 3 * \pi/4) = -1 & \text{if } n \in \mathcal{Z} \text{ and } |n * 2 * \pi + 3 * \pi/4| \le big\_angle\_r_F \\
cot^*_F(-x) = -cot^*_F(x) &
\end{array}
$$

The cot_F operation:

$cot_F : F \to F \cup \{\mathbf{underflow}, \mathbf{floating\_overflow}, \mathbf{pole}, \mathbf{angle\_too\_big}\}$

$$
\begin{array}{lll}
cot_F(x) & = trans\_result_F(cot^*_F(x)) & \text{if } x \in F \text{ and } x \ne 0 \text{ and } |x| \le big\_angle\_r_F \\
& = \mathbf{pole}(+\infty) & \text{if } x = 0 \\
& = \mathbf{pole}(-\infty) & \text{if } x = \mathbf{-0} \\
& = \mathbf{angle\_too\_big(qNaN)} & \text{if } |x| > big\_angle\_r_F \\
& = \mathbf{invalid(qNaN)} & \text{if } x \in \{-\infty, +\infty\} \\
& = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
& = \mathbf{invalid(qNaN)} & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

5.3.6.7 Radian secantus operation

The sec^*_F approximation helper function:

$sec^*_F : \mathcal{R} \to \mathcal{R}$

$sec^*_F(x)$ returns a close approximation to $\sec(x)$ in $\mathcal{R}$ if $|x| \le big\_angle\_r_F$, with maximum error $max\_error\_tan_F$.

Further requirements on the sec^*_F approximation helper function:

$$
\begin{array}{ll}
sec^*_F(n * 2 * \pi) = 1 & \text{if } n \in \mathcal{Z} \text{ and } |n * 2 * \pi| \le big\_angle\_r_F \\
sec^*_F(n * 2 * \pi + \pi/3) = 2 & \text{if } n \in \mathcal{Z} \text{ and } |n * 2 * \pi + \pi/3| \le big\_angle\_r_F \\
sec^*_F(n * 2 * \pi + 2 * \pi/3) = -2 & \text{if } n \in \mathcal{Z} \text{ and } |n * 2 * \pi + 2 * \pi/3| \le big\_angle\_r_F \\
sec^*_F(n * 2 * \pi + \pi) = -1 & \text{if } n \in \mathcal{Z} \text{ and } |n * 2 * \pi + \pi| \le big\_angle\_r_F \\
sec^*_F(x) = 1 & \text{if } sec^*_F(x) \ne \sec(x) \text{ and } |x| < \sqrt{epsilon_F} \\
sec^*_F(-x) = sec^*_F(x) &
\end{array}
$$

The sec_F operation:

$sec_F : F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{angle\_too\_big}\}$

$sec_F(x) = trans\_result_F(sec^*_F(x))$ if $x \in F$ and $|x| \leq big\_angle\_r_F$
$= 1$ if $x = \mathbf{-0}$
$= \mathbf{angle\_too\_big(qNaN)}$ if $|x| > big\_angle\_r_F$
$= \mathbf{invalid(qNaN)}$ if $x \in \{-\infty, +\infty\}$
$= \mathbf{qNaN}$ if x is a quiet NaN
$= \mathbf{invalid(qNaN)}$ if x is a signalling NaN

5.3.6.8 Radian cosecantus operation

The csc^*_F approximation helper function:

$csc^*_F : \mathcal{R} \to \mathcal{R}$

$csc^*_F(x)$ returns a close approximation to $\csc(x)$ in $\mathcal{R}$ if $|x| \leq big\_angle\_r_F$, with maximum error $max\_error\_tan_F$.

Further requirements on the csc^*_F approximation helper function:

$csc^*_F(n * 2 * \pi + \pi/6) = 2$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi/6| \leq big\_angle\_r_F$
$csc^*_F(n * 2 * \pi + \pi/2) = 1$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi/2| \leq big\_angle\_r_F$
$csc^*_F(n * 2 * \pi + 5 * \pi/6) = 2$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + 5 * \pi/6| \leq big\_angle\_r_F$
$csc^*_F(-x) = -csc^*_F(x)$

The csc_F operation:

$csc_F : F \to F \cup \{\mathbf{floating\_overflow}, \mathbf{pole}, \mathbf{angle\_too\_big}\}$

$csc_F(x) = trans\_result_F(csc^*_F(x))$ if $x \in F$ and $x \neq 0$ and $|x| \leq big\_angle\_r_F$
$= \mathbf{pole}(+\infty)$ if $x = 0$
$= \mathbf{pole}(-\infty)$ if $x = \mathbf{-0}$
$= \mathbf{angle\_too\_big}$ if $|x| > big\_angle\_r_F$
$= \mathbf{invalid(qNaN)}$ if $x \in \{-\infty, +\infty\}$
$= \mathbf{qNaN}$ if x is a quiet NaN
$= \mathbf{invalid(qNaN)}$ if x is a signalling NaN

5.3.6.9 Radian arcus sinus operation

The $arcsin^*_F$ approximation helper function:

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

$arcsin^*_F(x)$ 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:

$arcsin^*_F(1/2) = \pi/6$
$arcsin^*_F(1) = \pi/2$
$arcsin^*_F(x) = x$ if $arcsin^*_F(x) \neq \arcsin(x)$ and $|x| < \sqrt{2 * epsilon_F / r_F}$
$arcsin^*_F(-x) = -arcsin^*_F(x)$

The $arcsin_F$ operation:

$arcsin_F : F \to F \cup \{\mathbf{invalid}\}$

$arcsin_F(x) = trans\_result_F(arcsin^*_F(x))$
if $x \in F$ and $fminN_F < |x| \leq 1$
$= x$ if $x \in F$ and $|x| \leq fminN_F$
$= \mathbf{-0}$ if $x = \mathbf{-0}$
$= \mathbf{invalid(qNaN)}$ if $x \in F$ and ($x < -1$ or $x > 1$)
$= \mathbf{invalid(qNaN)}$ if $x \in \{-\infty, +\infty\}$
$= \mathbf{qNaN}$ if x is a quiet NaN

$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN

NOTE – **underflow** is explicitly avoided.

5.3.6.10 Radian arcus cosinus operation

The $arccos^*_F$ approximation helper function:

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

$arccos^*_F(x)$ 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:

$arccos^*_F(1/2) = \pi/3$
$arccos^*_F(0) = \pi/2$
$arccos^*_F(-1/2) = 2 * \pi/3$
$arccos^*_F(-1) = \pi$

The $arccos_F$ operation:

$arccos_F : F \to F \cup \{\mathbf{invalid}\}$

$arccos_F(x) = trans\_result_F(arccos^*_F(x))$ if $x \in F$ and $-1 \le x \le 1$
$= arccos_F(0)$ if $x = \mathbf{-0}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $x \in F$ and $(x < -1$ or $x > 1)$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $x \in \{-\infty, +\infty\}$
$= \mathbf{qNaN}$ if x is a quiet NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN

5.3.6.11 Radian arcus operation

The arc^*_F approximation helper function:

$arc^*_F : F \times F \to \mathcal{R}$

$arc^*_F(x, y)$ returns a close approximation to $arc(x, y)$ in $\mathcal{R}$, with maximum error $max\_error\_tan_F$.

NOTES

1 The mathematical arc function is defined in section 4.

2 The arc operations are often called `arctan2` (with the co-ordinate arguments swapped), or `arccot2`.

Further requirements on the arc^*_F approximation helper function:

$arc^*_F(x, 0) = 0$ if $x > 0$
$arc^*_F(x, x) = \pi/4$ if $x > 0$
$arc^*_F(0, y) = \pi/2$ if $y > 0$
$arc^*_F(x, -x) = 3 * \pi/4$ if $x < 0$
$arc^*_F(x, 0) = \pi$ if $x < 0$
$arc^*_F(x, -y) = -arc^*_F(x, y)$ if $y \neq 0$ or $x > 0$

The arc_F operation:

$arc_F : F \times F \to F \cup \{\mathbf{underflow}, \mathbf{invalid}\}$

$$
\begin{array}{lll}
arc_F(x,y) & = trans\_result_F(arc^*_F(x,y)) & \\
 & & \text{if } x,y \in F \text{ and } (x \neq 0 \text{ or } y \neq 0) \\
 & = \mathbf{invalid}(0) & \text{if } x = 0 \text{ and } y = 0 \\
 & = arc_F(0,y) & \text{if } x = \mathbf{-0} \text{ and } y \in F \cup \{-\infty, \mathbf{-0}, +\infty\} \\
 & = neg_F(arc_F(x,0)) & \text{if } y = \mathbf{-0} \text{ and } x \in F \cup \{-\infty, +\infty\} \\
 & & \\
 & = 0 & \text{if } x = +\infty \text{ and } y \in F \text{ and } y \geq 0 \\
 & = neg_F(0) & \text{if } x = +\infty \text{ and } y \in F \text{ and } y < 0 \\
 & = nearest_F(\pi/4)?\text{inval}? & \text{if } x = +\infty \text{ and } y = +\infty \\
 & = nearest_F(\pi/2) & \text{if } x \in F \text{ and } y = +\infty \\
 & = nearest_F(3*\pi/4)?\text{inval}? & \text{if } x = -\infty \text{ and } y = +\infty \\
 & = nearest_F(\pi) & \text{if } x = -\infty \text{ and } y \in F \text{ and } y \geq 0 \\
 & = nearest_F(-\pi) & \text{if } x = -\infty \text{ and } y \in F \text{ and } y < 0 \\
 & = nearest_F(-3*\pi/4)?\text{inval}? & \text{if } x = -\infty \text{ and } y = -\infty \\
 & = nearest_F(-\pi/2) & \text{if } x \in F \text{ and } y = -\infty \\
 & = nearest_F(-\pi/4)?\text{inval}? & \text{if } x = +\infty \text{ and } y = -\infty \\
 & & \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN and } y \text{ is not a signalling NaN} \\
 & = \mathbf{qNaN} & \text{if } y \text{ is a quiet NaN and } x \text{ is not a signalling NaN} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN or } y \text{ is a signalling NaN}
\end{array}
$$

5.3.6.12 Radian arcus tangentus operation

The $arctan^*_F$ approximation helper function:

$arctan^*_F : F \to \mathcal{R}$

$arctan^*_F(x)$ 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:

$$
\begin{array}{ll}
arctan^*_F(1) = \pi/4 & \\
arctan^*_F(x) = x & \text{if } arctan^*_F(x) \neq \arctan(x) \text{ and } |x| \leq \sqrt{1.5 * epsilon_F / r_F} \\
arctan^*_F(x) = \pi/2 & \text{if } arctan^*_F(x) \neq \arctan(x) \text{ and } x > 3 * r_F / epsilon_F \\
arctan^*_F(-x) = -arctan^*_F(x) &
\end{array}
$$

The $arctan_F$ operation:

$arctan_F : F \to F$

$$
\begin{array}{lll}
arctan_F(x) & = trans\_result_F(arctan^*_F(x)) & \\
 & & \text{if } x \in F \text{ and } fminN_F < |x| \\
 & = x & \text{if } x \in F \text{ and } |x| \leq fminN_F \\
 & = \mathbf{-0} & \text{if } x = \mathbf{-0} \\
 & = trans\_result_F(-\pi/2) & \text{if } x = -\infty \\
 & = trans\_result_F(\pi/2) & \text{if } x = +\infty \\
 & = \mathbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \mathbf{invalid}(\mathbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTES

1 $arctan_F(x) \approx arc_F(1,x)$

2 **underflow** is explicitly avoided.

5.3.6.13 Radian arcus cotangentus operation

The $arccot^*_F$ and $arcctg^*_F$ approximation helper functions:

$arccot^*_F : F \to \mathcal{R}$
$arcctg^*_F : F \to \mathcal{R}$

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

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

Further requirements on the $arccot^*_F$ and $arcctg^*_F$ approximation helper functions:

$arccot^*_F(1) = \pi/4$
$arccot^*_F(0) = \pi/2$
$arccot^*_F(-1) = 3 * \pi/4$
$arccot^*_F(x) = \pi$ if $arccot^*_F(x) \neq \text{arccot}(x)$ and $x < -3 * r_F/epsilon_F$

$arcctg^*_F(x) = arccot^*_F(x)$ if $x \geq 0$
$arcctg^*_F(-x) = -arcctg^*_F(x)$

The $arccot_F$ operation:

$arccot_F : F \to F \cup \{\textbf{underflow}\}$

$$
\begin{array}{lll}
arccot_F(x) & = trans\_result_F(arccot^*_F(x)) & \\
 & & \text{if } x \in F \\
 & = trans\_result_F(\pi/2) & \text{if } x = \textbf{-0} \\
 & = trans\_result_F(\pi) & \text{if } x = -\infty \\
 & = 0 & \text{if } x = +\infty \\
 & = \textbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTES

1 $arccot_F(x) \approx arc_F(x, 1)$.

2 There is no "jump" at zero for $arccot_F$.

The $arcctg_F$ operation:

$arcctg_F : F \to F \cup \{\textbf{underflow}\}$

$$
\begin{array}{lll}
arcctg_F(x) & = trans\_result_F(arcctg^*_F(x)) & \\
 & & \text{if } x \in F \\
 & = trans\_result_F(-\pi/2) & \text{if } x = \textbf{-0} \\
 & = \textbf{-0} & \text{if } x = -\infty \\
 & = 0 & \text{if } x = +\infty \\
 & = \textbf{qNaN} & \text{if } x \text{ is a quiet NaN} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } x \text{ is a signalling NaN}
\end{array}
$$

NOTE 3 – $arcctg_F(neg_F(x)) = neg_F(arcctg_F(x))$.

5.3.6.14 Radian arcus secantus operation

The $arcsec^*_F$ approximation helper function:

$arcsec^*_F : F \to \mathcal{R}$

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

Further requirements on the $arcsec^*_F$ approximation helper function:

$arcsec_F^*(2) = \pi/3$
$arcsec_F^*(-2) = 2 * \pi/3$
$arcsec_F^*(-1) = \pi$
$arcsec_F^*(x) \le \pi/2$ if $x > 0$
$arcsec_F^*(x) \ge \pi/2$ if $x < 0$
$arcsec_F^*(x) = \pi/2$ if $arcsec_F^*(x) \neq \text{arcsec}(x)$ and $|x| > 3 * r_F/epsilon_F$

The $arcsec_F$ operation:

$arcsec_F : F \to F \cup \{\textbf{invalid}\}$

$arcsec_F(x) = trans\_result_F(arcsec_F^*(x))$ if $x \in F$ and $(x \le -1$ or $x \ge 1)$
$= \textbf{invalid}(\textbf{qNaN})$ if $x \in F$ and $-1 < x < 1$
$= \textbf{invalid}(\textbf{qNaN})$ if $x = \textbf{-0}$
$= trans\_result_F(\pi/2)$ if $x = -\infty$
$= trans\_result_F(\pi/2)$ if $x = +\infty$
$= \textbf{qNaN}$ if x is a quiet NaN
$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN

5.3.6.15 Radian arcus cosecantus operation

The $arccsc_F^*$ approximation helper function:

$arccsc_F^* : F \to \mathcal{R}$

$arccsc_F^*(x)$ returns a close approximation to $\text{arccsc}(x)$ in $\mathcal{R}$, with maximum error $max\_error\_tan_F$.

Further requirements on the $arccsc_F^*$ approximation helper function:

$arccsc_F^*(2) = \pi/6$
$arccsc_F^*(1) = \pi/2$
$arccsc_F^*(-x) = -arccsc_F^*(x)$

The $arccsc_F$ operation:

$arccsc_F : F \to F \cup \{\textbf{underflow}, \textbf{invalid}\}$

$arccsc_F(x) = trans\_result_F(arccsc_F^*(x))$ if $x \in F$ and $|x| \ge 1$
$= \textbf{invalid}(\textbf{qNaN})$ if $x \in F$ and $-1 < x < 1$
$= \textbf{invalid}(\textbf{qNaN})$ if $x = \textbf{-0}$
$= \textbf{-0}$ if $x = -\infty$
$= 0$ if $x = +\infty$
$= \textbf{qNaN}$ if x is a quiet NaN
$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN

5.3.7 Operations for argument angular-unit trigonometrics and inverse argument angular-unit trigonometrics

There shall be one big-angle parameter for argument angular-unit trigonometric operations:

$big\_angle\_u_F \in F$

It is required to have the following default value:

$big\_angle\_u_F = \lceil r_F^{\lceil p_F/2 \rceil}/6 \rceil$

NOTE 1 – The user may be allowed to narrow this value, but should not be allowed to widen it beyond the value given here.

There shall be one derived parameter signifying the minimum allowed angular unit:

$min\_angular\_unit_F = r_F * fminN_F/epsilon_F = r_F^{(emin_F - 1 + p_F)}$

It is specified for two reasons. Firstly, if the type F has no denormal values ($denorm_F = \mathbf{false}$), some angle values in F are not representable after normalisation if the angular unit is too small (this gives the firm limit above). Secondly, even if F has denormal values ($denorm_F = \mathbf{true}$), very tiny angular units do not allow the representable angles to be particularly dense, not even if the angular value is within the first cycle. This does in itself not give rise to a particular limit value, but the limit value defined here is reasonable.

To make the requirements a bit easier to express, let $G_F = \{x \in F \mid |x| \geq min\_angular\_unit_F\}$.

NOTE 2 – Negative angular units have not been included since this simplifies the specification of the inverse trigonometric argument angular unit operations somewhat, and the exclusion is not judged to be significant.

There shall be two parameterised maximum error parameters for angular-unit argument trigonometric operations.

$max\_error\_sinu_F : F \to F \cup \{\mathbf{invalid}\}$
$max\_error\_tanu_F : F \to F \cup \{\mathbf{invalid}\}$

Let $T = \{1, 2, 360, 400, 6400\}$. T consists of angle values for exactly one revolution for some common non-radian angular units: cycles, half-cycles, arc degrees, grades, and mils.

For $u \in G_F$, the $max\_error\_sinu_F(u)$ parameter shall be in the interval $[max\_error\_sin_F, 2]$. The $max\_error\_sinu_F(u)$ parameter shall be equal to $max\_error\_sin_F$ if $u \in T$.

For $u \in G_F$, the $max\_error\_tanu_F(u)$ parameter shall be in the interval $[max\_error\_tan_F, 4]$. The $max\_error\_tanu_F(u)$ parameter shall be equal to $max\_error\_tan_F$ if $u \in T$.

The $max\_error\_sinu_F(u)$ and $max\_error\_tanu_F(u)$ parameters return **invalid** if $u \notin G_F$.

All of the argument angular unit trigonometric, and argument angular unit inverse trigonometric, approximation helper functions, including those for normalisation, angular unit conversion, and arc, are exempted from the monotonicity requirement for the angular unit argument.

5.3.7.1 Argument angular-unit angle normalisation operations

The argument angular-unit normalisation computes exactly $rad(2 * \pi * x/u) * u/(2 * \pi)$, where x is the angular value, and u is the angular unit.

The $cycle_F$ operation:

$cycle_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{angle\_too\_big}, \mathbf{invalid}\}$

$cycle_F(u, x)$	$= x - (\text{round}(x/u) * u)$	if $u \in G_F$ and $x \in F$ and $(x \geq 0$ or $x - (\text{round}(x/u) * u) \neq 0)$ and $\lvert x/u \rvert \leq big\_angle\_u_F$
	$= \mathbf{-0}$	if $u \in G_F$ and $x \in F$ and $x < 0$ and $x - (\text{round}(x/u) * u) = 0$ and $\lvert x/u \rvert \leq big\_angle\_u_F$
	$= \mathbf{-0}$	if $u \in G_F$ and $x = \mathbf{-0}$
	$= \mathbf{angle\_too\_big}(\mathbf{qNaN})$	if $u \in G_F$ and $x \in F$ and $\lvert x/u \rvert > big\_angle\_u_F$
	$= \mathbf{invalid}(\mathbf{qNaN})$	if $u \in F$ and $\lvert u \rvert < min\_angular\_unit_F$ and $x \in F \cup \{\mathbf{-0}\}$
	$= \mathbf{invalid}(\mathbf{qNaN})$	if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
	$= \mathbf{invalid}(\mathbf{qNaN})$	if $u \in F$ and $x \in \{-\infty, +\infty\}$
	$= \mathbf{qNaN}$	if x is a quiet NaN and u is not a signalling NaN
	$= \mathbf{qNaN}$	if u is a quiet NaN and x is not a signalling NaN

$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or u is a signalling NaN

The $axis\_cycle_F$ operation:

$axis\_cycle_F : F \times F \to ((F \times F) \times (F \cup \{\mathbf{-0}\})) \cup \{\mathbf{angle\_too\_big}, \mathbf{invalid}\}$

$axis\_cycle_F(u, x)$

$= (axis(u,x), result_F(x - ((round(x/(u/4)) * u/4), rnd_F))$ if $u \in G_F$ and $x \in F$ and $(x/u \geq 0$ or $x - ((round(x * 4/u) * u/4)) \neq 0)$ and $|x/u| \leq big\_angle\_u_F$

$= (axis(u,x), \mathbf{-0})$ if $u \in G_F$ and $x \in F$ and $x/u < 0$ and $x - (round(x * 4/u) * u/4) = 0$ and $|x/u| \leq big\_angle\_u_F$

$= ((1,0), \mathbf{-0})$ if $u \in G_F$ and $x = \mathbf{-0}$ and $u > 0$

$= ((1,0), 0)$ if $u \in G_F$ and $x = \mathbf{-0}$ and $u < 0$

$= \mathbf{angle\_too\_big}((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN})$ if $u \in G_F$ and $x \in F$ and $|x/u| > big\_angle\_u_F$

$= \mathbf{invalid}((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN})$ if $u \in F$ and $|u| < min\_angular\_unit_F$ and $x \in F \cup \{\mathbf{-0}\}$

$= \mathbf{invalid}((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN})$ if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{invalid}((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN})$ if $u \in F$ and $x \in \{-\infty, +\infty\}$

$= ((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN})$ if x is a quiet NaN and u is not a signalling NaN

$= ((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN})$ if u is a quiet NaN and x is not a signalling NaN

$= \mathbf{invalid}((\mathbf{qNaN}, \mathbf{qNaN}), \mathbf{qNaN})$ if x is a signalling NaN or u is a signalling NaN

where

$axis(u,x) = (1,0)$ if $\text{round}(x * 4/u) \bmod 4 = 0$

$= (0,1)$ if $\text{round}(x * 4/u) \bmod 4 = 1$

$= (-1,0)$ if $\text{round}(x * 4/u) \bmod 4 = 2$

$= (0,-1)$ if $\text{round}(x * 4/u) \bmod 4 = 3$

NOTES

1 $axis\_cycle_F(u,x)$ is exact when $div_F(u,4) = u/4$.

2 $cycle_F$ is an exact operation.

3 $cycle_F(u,x)$ has a result in the interval $[-u/2, u/2]$ if $u > 0$.

4 A zero resulting angle is negative if the original angle value is negative.

5 The $cycle_F$ operation is used also in the specifications of the unit argument trigonometric operations.

5.3.7.2 Argument angular-unit sinus operation

The $sinu^*_F$ approximation helper function:

$sinu^*_F : F \times \mathcal{R} \to \mathcal{R}$

$sinu^*_F(u,x)$ returns a close approximation to $\sin(x*2*\pi/u)$ in $\mathcal{R}$ if $u \neq 0$, with maximum error $max\_error\_sinu_F(u)$.

Further requirements on the $sinu^*_F$ approximation helper function:

$sinu^*_F(u, n*u+x) = sinu^*_F(u,x)$	if $n \in \mathcal{Z}$ and $u \in F$ and $u \neq 0$
$sinu^*_F(u, u/12) = 1/2$	if $u \in F$ and $u \neq 0$
$sinu^*_F(u, u/4) = 1$	if $u \in F$ and $u \neq 0$
$sinu^*_F(u, 5*u/12) = 1/2$	if $u \in F$ and $u \neq 0$
$sinu^*_F(u, -x) = -sinu^*_F(u,x)$	if $u \in F$ and $u \neq 0$
$sinu^*_F(-u, x) = -sinu^*_F(u,x)$	if $u \in F$ and $u \neq 0$

NOTE – $sinu^*_F(u,x) \approx x*2*\pi/u$ if $|x*2*\pi/u| < fminN_F$.

The $sinu_F$ operation:

$sinu_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{invalid}, \mathbf{angle\_too\_big}\}$

$sinu_F(u,x) = trans\_result_F(sinu^*_F(u,x))$	if $cycle_F(u,x) \in F$ and $cycle_F(u,x) \neq -u/2$
$= \mathbf{-0}$	if $cycle_F(u,x) \in F$ and $cycle_F(u,x) = -u/2$
$= \mathbf{-0}$	if $cycle_F(u,x) = \mathbf{-0}$
$= \mathbf{angle\_too\_big}(\mathbf{qNaN})$	if $u \in G_F$ and $x \in F$ and $\lvert x/u \rvert > big\_angle\_u_F$
$= \mathbf{invalid}(\mathbf{qNaN})$	if $u \in F$ and $\lvert u \rvert < min\_angular\_unit_F$ and $x \in F \cup \{\mathbf{-0}\}$
$= \mathbf{invalid}(\mathbf{qNaN})$	if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})$	if $u \in F$ and $x \in \{-\infty, +\infty\}$
$= \mathbf{qNaN}$	if x is a quiet NaN and u is not a signalling NaN
$= \mathbf{qNaN}$	if u is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$	if x is a signalling NaN or u is a signalling NaN

5.3.7.3 Argument angular-unit cosinus operation

The $cosu^*_F$ approximation helper function:

$cosu^*_F : F \times \mathcal{R} \to \mathcal{R}$

$cosu^*_F(u,x)$ returns a close approximation to $\cos(x*2*\pi/u)$ in $\mathcal{R}$ if $u \neq 0$, with maximum error $max\_error\_sinu_F(u)$.

Further requirements on the $cosu^*_F$ approximation helper function:

$cosu^*_F(u, n*u+x) = cosu^*_F(u,x)$	if $n \in \mathcal{Z}$ and $u \in F$ and $u \neq 0$
$cosu^*_F(u, 0) = 1$	if $u \in F$ and $u \neq 0$
$cosu^*_F(u, u/6) = 1/2$	if $u \in F$ and $u \neq 0$
$cosu^*_F(u, u/3) = -1/2$	if $u \in F$ and $u \neq 0$
$cosu^*_F(u, u/2) = -1$	if $u \in F$ and $u \neq 0$
$cosu^*_F(u, -x) = cosu^*_F(u,x)$	if $u \in F$ and $u \neq 0$
$cosu^*_F(-u, x) = cosu^*_F(u,x)$	if $u \in F$ and $u \neq 0$

NOTE – $cosu^*_F(u,x) = 1$ should hold if $|x*2*\pi/u| < \sqrt{epsilon_F/r_F}$

The $cosu_F$ operation:

$cosu_F : F \times F \to F \cup \{\mathbf{underflow}, \mathbf{invalid}, \mathbf{angle\_too\_big}\}$

$cosu_F(u,x) = trans\_result_F(cosu^*_F(u,x))$	if $cycle_F(u,x) \in F$
$= 1$	if $cycle_F(u,x) = \mathbf{-0}$

$= \mathbf{angle\_too\_big}(\mathbf{qNaN})$ if $u \in G_F$ and $x \in F$ and $|x/u| > big\_angle\_u_F$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in F$ and $|u| < min\_angular\_unit_F$ and $x \in F \cup \{\mathbf{-0}\}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in F$ and $x \in \{-\infty, +\infty\}$

$= \mathbf{qNaN}$ if x is a quiet NaN and u is not a signalling NaN
$= \mathbf{qNaN}$ if u is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or u is a signalling NaN

5.3.7.4 Argument angular-unit cosinus with sinus operation

$cossinu_F : F \times F \to (F \times (F \cup \{\mathbf{-0}\})) \cup \{\mathbf{underflow}, \mathbf{invalid}, \mathbf{angle\_too\_big}\}$
$cossinu_F(u, x) = (cosu_F(u, x), sinu_F(u, x))$

5.3.7.5 Argument angular-unit tangentus operation

The $tanu^*_F$ approximation helper function:

$tanu^*_F : F \times \mathcal{R} \to \mathcal{R}$

$tanu^*_F(u, x)$ returns a close approximation to $\tan(x * 2 * \pi/u)$ in $\mathcal{R}$ if $u \neq 0$, with maximum error $max\_error\_tanu_F(u)$.

Further requirements on the $tanu^*_F$ approximation helper function:

$tanu^*_F(u, n * u + x) = tanu^*_F(u, x)$ if $n \in \mathcal{Z}$ and $u \in F$ and $u \neq 0$
$tanu^*_F(u, u/8) = 1$ if $u \in F$ and $u \neq 0$
$tanu^*_F(u, 3 * u/8) = -1$ if $u \in F$ and $u \neq 0$
$tanu^*_F(u, -x) = -tanu^*_F(u, x)$ if $u \in F$ and $u \neq 0$
$tanu^*_F(-u, x) = -tanu^*_F(u, x)$ if $u \in F$ and $u \neq 0$

NOTE 1 – $tanu^*_F(u, x) \approx x * 2 * \pi/u$ if $|x * 2 * \pi/u| < fminN_F$.

The $tanu_F$ operation:

$tanu_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{pole}, \mathbf{floating\_overflow}, \mathbf{underflow}, \mathbf{invalid}, \mathbf{angle\_too\_big}\}$

$tanu_F(u, x) = trans\_result_F(tanu^*_F(u, x))$ if $cycle_F(u, x) \in F$ and $cycle_F(u, x) \notin \{-u/2, -u/4, u/4\}$
$= \mathbf{-0}$ if $cycle_F(u, x) \in F$ and $cycle_F(u, x) = -u/2$
$= \mathbf{-0}$ if $cycle_F(u, x) = \mathbf{-0}$
$= \mathbf{pole}(+\infty)$ if $cycle_F(u, x) = u/4$
$= \mathbf{pole}(-\infty)$ if $cycle_F(u, x) = -u/4$

$= \mathbf{angle\_too\_big}(\mathbf{qNaN})$ if $u \in G_F$ and $x \in F$ and $|x/u| > big\_angle\_u_F$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in F$ and $|u| < min\_angular\_unit_F$ and $x \in F \cup \{\mathbf{-0}\}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in F$ and $x \in \{-\infty, +\infty\}$

$= \mathbf{qNaN}$ if x is a quiet NaN and u is not a signalling NaN
$= \mathbf{qNaN}$ if u is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or u is a signalling NaN

NOTE 2 – The **pole** notification can arise for $tanu_F(u, x)$ only when $u/4$ is in F.

5.3.7.6 Argument angular-unit cotangentus operation

The $cotu^*_F$ approximation helper function:

$cotu^*_F : F \times \mathcal{R} \to \mathcal{R}$

$cotu^*_F(u, x)$ returns a close approximation to $\cot(x * 2 * \pi / u)$ in $\mathcal{R}$ if $u \neq 0$, with maximum error $max\_error\_tanu_F(u)$.

Further requirements on the $cotu^*_F$ approximation helper function:

$cotu^*_F(u, n * u + x) = cotu^*_F(u, x)$	if $n \in \mathcal{Z}$ and $u \in F$ and $u \neq 0$
$cotu^*_F(u, u/8) = 1$	if $u \in F$ and $u \neq 0$
$cotu^*_F(u, 3 * u/8) = -1$	if $u \in F$ and $u \neq 0$
$cotu^*_F(u, -x) = -cotu^*_F(u, x)$	if $u \in F$ and $u \neq 0$
$cotu^*_F(-u, x) = -cotu^*_F(u, x)$	if $u \in F$ and $u \neq 0$

The $cotu_F$ operation:

$cotu_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{pole}, \mathbf{floating\_overflow}, \mathbf{underflow}, \mathbf{invalid}, \mathbf{angle\_too\_big}\}$

$cotu_F(u, x)$	$= trans\_result_F(cotu^*_F(u, x))$	if $cycle_F(u, x) \in F$ and $cycle_F(u, x) \notin \{-u/2, -u/4, 0, u/2\}$
	$= \mathbf{-0}$	if $cycle_F(u, x) \in F$ and $cycle_F(u, x) = -u/4$
	$= \mathbf{pole}(+\infty)$	if $cycle_F(u, x) = 0$
	$= \mathbf{pole}(-\infty)$	if $cycle_F(u, x) = \mathbf{-0}$
	$= \mathbf{pole}(+\infty)$	if $cycle_F(u, x) = u/2$
	$= \mathbf{pole}(-\infty)$	if $cycle_F(u, x) = -u/2$
	$= \mathbf{angle\_too\_big}(\mathbf{qNaN})$	if $u \in G_F$ and $x \in F$ and $\lvert x/u \rvert > big\_angle\_u_F$
	$= \mathbf{invalid}(\mathbf{qNaN})$	if $u \in F$ and $\lvert u \rvert < min\_angular\_unit_F$ and $x \in F \cup \{\mathbf{-0}\}$
	$= \mathbf{invalid}(\mathbf{qNaN})$	if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
	$= \mathbf{invalid}(\mathbf{qNaN})$	if $u \in F$ and $x \in \{-\infty, +\infty\}$
	$= \mathbf{qNaN}$	if x is a quiet NaN and u is not a signalling NaN
	$= \mathbf{qNaN}$	if u is a quiet NaN and x is not a signalling NaN
	$= \mathbf{invalid}(\mathbf{qNaN})$	if x is a signalling NaN or u is a signalling NaN

5.3.7.7 Argument angular-unit secantus operation

The $secu^*_F$ approximation helper function:

$secu^*_F : F \times \mathcal{R} \to \mathcal{R}$

$secu^*_F(u, x)$ returns a close approximation to $\sec(x * 2 * \pi / u)$ in $\mathcal{R}$ if $u \neq 0$, with maximum error $max\_error\_tanu_F(u)$.

Further requirements on the $secu^*_F$ approximation helper function:

$secu^*_F(u, n * u + x) = secu^*_F(u, x)$	if $n \in \mathcal{Z}$ and $u \in F$ and $u \neq 0$
$secu^*_F(u, 0) = 1$	if $u \in F$ and $u \neq 0$
$secu^*_F(u, u/6) = 2$	if $u \in F$ and $u \neq 0$
$secu^*_F(u, u/3) = -2$	if $u \in F$ and $u \neq 0$
$secu^*_F(u, u/2) = -1$	if $u \in F$ and $u \neq 0$
$secu^*_F(u, -x) = secu^*_F(u, x)$	if $u \in F$ and $u \neq 0$
$secu^*_F(-u, x) = secu^*_F(u, x)$	if $u \in F$ and $u \neq 0$
$secu^*_F(u, x) = 1$	if $\lvert x * 2 * \pi / u \rvert < 0.5 * \sqrt{epsilon_F}$

The $secu_F$ operation:

$secu_F : F \times F \to F \cup \{\textbf{pole}, \textbf{floating\_overflow}, \textbf{invalid}, \textbf{angle\_too\_big}\}$

$$
\begin{array}{lll}
secu_F(u,x) & = trans\_result_F(secu^*_F(u,x)) & \\
 & & \text{if } cycle_F(u,x) \in F \text{ and } cycle_F(u,x) \notin \{-u/4, u/4\} \\
 & = 1 & \text{if } cycle_F(u,x) = \textbf{-0} \\
 & = \textbf{pole}(+\infty) & \text{if } cycle_F(u,x) = u/4 \\
 & = \textbf{pole}(+\infty) & \text{if } cycle_F(u,x) = -u/4 \\
 & & \\
 & = \textbf{angle\_too\_big}(\textbf{qNaN}) & \text{if } u \in G_F \text{ and } x \in F \text{ and } |x/u| > big\_angle\_u_F \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } u \in F \text{ and } |u| < min\_angular\_unit_F \text{ and } x \in F \cup \{\textbf{-0}\} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } u \in \{-\infty, \textbf{-0}, +\infty\} \text{ and } x \in F \cup \{-\infty, \textbf{-0}, +\infty\} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } u \in F \text{ and } x \in \{-\infty, +\infty\} \\
 & & \\
 & = \textbf{qNaN} & \text{if } x \text{ is a quiet NaN and } u \text{ is not a signalling NaN} \\
 & = \textbf{qNaN} & \text{if } u \text{ is a quiet NaN and } x \text{ is not a signalling NaN} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } x \text{ is a signalling NaN or } u \text{ is a signalling NaN}
\end{array}
$$

5.3.7.8 Argument angular-unit cosecantus operation

The $cscu^*_F$ approximation helper function:

$cscu^*_F : F \times \mathcal{R} \to \mathcal{R}$

$cscu^*_F(u,x)$ returns a close approximation to $\csc(x * 2 * \pi/u)$ in $\mathcal{R}$ if $u \neq 0$, with maximum error $max\_error\_tanu_F(u)$.

Further requirements on the $cscu^*_F$ approximation helper function:

$$
\begin{array}{ll}
cscu^*_F(u, n * u + x) = cscu^*_F(u,x) & \text{if } n \in \mathcal{Z} \text{ and } u \in F \text{ and } u \in 0 \\
cscu^*_F(u, u/12) = 2 & \text{if } u \in F \text{ and } u \neq 0 \\
cscu^*_F(u, u/4) = 1 & \text{if } u \in F \text{ and } u \neq 0 \\
cscu^*_F(u, 5 * u/12) = 2 & \text{if } u \in F \text{ and } u \neq 0 \\
cscu^*_F(u, -x) = -cscu^*_F(u,x) & \text{if } u \in F \text{ and } u \neq 0 \\
cscu^*_F(-u, x) = -cscu^*_F(u,x) & \text{if } u \in F \text{ and } u \neq 0
\end{array}
$$

The $cscu_F$ operation:

$cscu_F : F \times F \to F \cup \{\textbf{pole}, \textbf{floating\_overflow}, \textbf{invalid}, \textbf{angle\_too\_big}\}$

$$
\begin{array}{lll}
cscu_F(u,x) & = trans\_result_F(cscu^*_F(u,x)) & \\
 & & \text{if } cycle_F(u,x) \in F \text{ and } cycle_F(u,x) \notin \{-u/2, 0, u/2\} \\
 & = \textbf{pole}(+\infty) & \text{if } cycle_F(u,x) = 0 \\
 & = \textbf{pole}(-\infty) & \text{if } cycle_F(u,x) = \textbf{-0} \\
 & = \textbf{pole}(+\infty) & \text{if } cycle_F(u,x) = u/2 \\
 & = \textbf{pole}(-\infty) & \text{if } cycle_F(u,x) = -u/2 \\
 & & \\
 & = \textbf{angle\_too\_big}(\textbf{qNaN}) & \text{if } u \in G_F \text{ and } x \in F \text{ and } |x/u| > big\_angle\_u_F \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } u \in F \text{ and } |u| < min\_angular\_unit_F \text{ and } x \in F \cup \{\textbf{-0}\} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } u \in \{-\infty, \textbf{-0}, +\infty\} \text{ and } x \in F \cup \{-\infty, \textbf{-0}, +\infty\} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } u \in F \text{ and } x \in \{-\infty, +\infty\} \\
 & & \\
 & = \textbf{qNaN} & \text{if } x \text{ is a quiet NaN and } u \text{ is not a signalling NaN} \\
 & = \textbf{qNaN} & \text{if } u \text{ is a quiet NaN and } x \text{ is not a signalling NaN} \\
 & = \textbf{invalid}(\textbf{qNaN}) & \text{if } x \text{ is a signalling NaN or } u \text{ is a signalling NaN}
\end{array}
$$

5.3.7.9 Argument angular-unit arcus sinus operation

The $arcsinu^*_F$ approximation helper function:

$arcsinu^*_F : F \times F \to \mathcal{R}$

$arcsinu^*_F(u,x)$ returns a close approximation to $\arcsin(x) * u/(2 * \pi)$ in $\mathcal{R}$, with maximum error $max\_error\_sinu_F(u)$.

Further requirements on the $arcsinu^*_F$ approximation helper function:

$arcsinu^*_F(u, 1/2) = u/12$
$arcsinu^*_F(u, 1) = u/4$
$arcsinu^*_F(u, -x) = -arcsinu^*_F(u, x)$
$arcsinu^*_F(-u, x) = -arcsinu^*_F(u, x)$

NOTE – $arcsinu^*_F(u,x) \approx u/(2 * \pi)$ if $|x| < fminN_F$.

The $arcsinu_F$ operation:

$arcsinu_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{invalid}\}$

$arcsinu_F(u,x)$

$= trans\_result_F(arcsinu^*_F(u,x))$	if $u \in G_F$ and $x \in F$ and $\|x\| \le 1$ and $x \neq 0$
$= 0$	if $u \in G_F$ and $u > 0$ and $x = 0$
$= \mathbf{-0}$	if $u \in G_F$ and $u > 0$ and $x = \mathbf{-0}$
$= \mathbf{-0}$	if $u \in G_F$ and $u < 0$ and $x = 0$
$= 0$	if $u \in G_F$ and $u < 0$ and $x = \mathbf{-0}$
$= \mathbf{invalid(qNaN)}$	if $u \in G_F$ and $x \in F$ and $\|x\| > 1$
$= \mathbf{invalid(qNaN)}$	if $u \in G_F$ and $x \in \{-\infty, +\infty\}$
$= \mathbf{invalid(qNaN)}$	if $u \in F$ and $\|u\| < min\_angular\_unit_F$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{invalid(qNaN)}$	if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{qNaN}$	if x is a quiet NaN and u is not a signalling NaN
$= \mathbf{qNaN}$	if u is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid(qNaN)}$	if x is a signalling NaN or u is a signalling NaN

5.3.7.10 Argument angular-unit arcus cosinus operation

The $arccosu^*_F$ approximation helper function:

$arccosu^*_F : F \times F \to \mathcal{R}$

$arccosu^*_F(u,x)$ returns a close approximation to $\arccos(x) * u/(2 * \pi)$ in $\mathcal{R}$, with maximum error $max\_error\_sinu_F(u)$.

Further requirements on the $arccosu^*_F$ approximation helper function:

$arccosu^*_F(u, 1/2) = u/6$
$arccosu^*_F(u, 0) = u/4$
$arccosu^*_F(u, -1/2) = u/3$
$arccosu^*_F(u, -1) = u/2$
$arccosu^*_F(-u, x) = -arccosu^*_F(u, x)$

The $arccosu_F$ operation:

$arccosu_F : F \times F \to F \cup \{\mathbf{underflow}, \mathbf{invalid}\}$

$arccosu_F(u,x)$

$= trans\_result_F(arccosu^*_F(u,x))$ if $u \in G_F$ and $x \in F$ and $|x| \le 1$
$= trans\_result_F(u/4)$ if $u \in G_F$ and $x = \mathbf{-0}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in G_F$ and $x \in F$ and $|x| > 1$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in G_F$ and $x \in \{-\infty, +\infty\}$

$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in F$ and $|u| < min\_angular\_unit_F$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{qNaN}$ if x is a quiet NaN and u is not a signalling NaN
$= \mathbf{qNaN}$ if u is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or u is a signalling NaN

5.3.7.11 Argument angular-unit arcus operation

The $arcu^*_F$ approximation helper function:

$arcu^*_F : F \times F \times F \to \mathcal{R}$

$arcu^*_F(u,x,y)$ returns a close approximation to $arc(x,y) * u/(2*\pi)$ in $\mathcal{R}$, with maximum error $max\_error\_tanu_F(u)$.

Further requirements on the $arcu^*_F$ approximation helper function:

$arcu^*_F(u,x,x) = u/8$ if $x > 0$
$arcu^*_F(u,0,y) = u/4$ if $y > 0$
$arcu^*_F(u,x,-x) = 3*u/8$ if $x < 0$
$arcu^*_F(u,x,0) = u/2$ if $x < 0$
$arcu^*_F(u,x,-y) = -arcu^*_F(u,x,y)$ if $y \neq 0$ or $x > 0$
$arcu^*_F(-u,x,y) = -arcu^*_F(u,x,y)$

The $arcu_F$ operation:

$arcu_F : F \times F \times F \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{invalid}\}$

$arcu_F(u,x,y) = trans\_result_F(arcu^*_F(u,x,y))$ if $u \in G_F$ and $x, y \in F$ and ($x < 0$ or $y \neq 0$)
$= mul_F(u,0)$ if $u \in G_F$ and $x \in F$ and $x > 0$ and $y = 0$
$= \mathbf{invalid}(0)$ if $u \in G_F$ and $x = 0$ and $y = 0$
$= arcu_F(u,0,y)$ if $u \in G_F$ and $x = \mathbf{-0}$ and $y \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= neg_F(arcu_F(u,x,0))$ if $u \in G_F$ and $y = \mathbf{-0}$ and $x \in F \cup \{-\infty, +\infty\}$

$= mul_F(0,u)$ if $u \in G_F$ and $x = +\infty$ and $y \in F$ and $y \ge 0$
$= mul_F(neg_F(0),u)$ if $u \in G_F$ and $x = +\infty$ and $y \in F$ and $y < 0$
$= nearest_F(u/8)$?inval? if $u \in G_F$ and $x = +\infty$ and $y = +\infty$
$= nearest_F(u/4)$ if $u \in G_F$ and $x \in F$ and $y = +\infty$
$= nearest_F(3*u/8)$?inval? if $u \in G_F$ and $x = -\infty$ and $y = +\infty$
$= nearest_F(u/2)$ if $u \in G_F$ and $x = -\infty$ and $y \in F$ and $y \ge 0$
$= nearest_F(-u/2)$ if $u \in G_F$ and $x = -\infty$ and $y \in F$ and $y < 0$
$= nearest_F(-3*u/8)$?inval? if $u \in G_F$ and $x = -\infty$ and $y = -\infty$
$= nearest_F(-u/4)$ if $u \in G_F$ and $x \in F$ and $y = -\infty$
$= nearest_F(-u/8)$?inval? if $u \in G_F$ and $x = +\infty$ and $y = -\infty$

$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in F$ and $|u| < min\_angular\_unit_F$ and

$x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x, y \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{qNaN}$ if u is a quiet NaN and not x nor y is a signalling NaN

$= \mathbf{qNaN}$ if x is a quiet NaN and not u nor y is a signalling NaN

$= \mathbf{qNaN}$ if y is a quiet NaN and not u nor x is a signalling NaN

$= \mathbf{invalid}(\mathbf{qNaN})$ if u is a signalling NaN or x is a signalling NaN or y is a signalling NaN

5.3.7.12 Argument angular-unit arcus tangentus operation

The $arctanu_F^*$ approximation helper function:

$arctanu_F^* : F \times F \to \mathcal{R}$

$arctanu_F^*(u, x)$ returns a close approximation to $\arctan(x) * u/(2 * \pi)$ in $\mathcal{R}$, with maximum error $max\_error\_tanu_F(u)$.

Further requirements on the $arctanu_F^*$ approximation helper function:

$arctanu_F^*(u, 1) = u/8$

$arctanu_F^*(u, x) = u/4$ if $arctanu_F^*(u, x) \neq \arctan(x) * u/(2 * \pi)$ and $x > 3 * r_F/epsilon_F$

$arctanu_F^*(u, -x) = -arctanu_F^*(u, x)$

$arctanu_F^*(-u, x) = -arctanu_F^*(u, x)$

NOTE 1 – $arctanu_F^*(u, x) \approx u/(2 * \pi)$ if $|x| < fminN_F$

The $arctanu_F$ operation:

$arctanu_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{invalid}, \mathbf{underflow}\}$

$arctanu_F(u, x)$

$= trans\_result_F(arctanu_F^*(u, x))$ if $u \in G_F$ and $x \in F$ and $x \neq 0$

$= 0$ if $u \in G_F$ and $u > 0$ and $x = 0$

$= \mathbf{-0}$ if $u \in G_F$ and $u > 0$ and $x = \mathbf{-0}$

$= \mathbf{-0}$ if $u \in G_F$ and $u < 0$ and $x = 0$

$= 0$ if $u \in G_F$ and $u < 0$ and $x = \mathbf{-0}$

$= trans\_result_F(-u/4)$ if $u \in G_F$ and $x = -\infty$

$= trans\_result_F(u/4)$ if $u \in G_F$ and $x = +\infty$

$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in F$ and $|u| < min\_angular\_unit_F$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{invalid}(\mathbf{qNaN})$ if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{qNaN}$ if x is a quiet NaN and u is not a signalling NaN

$= \mathbf{qNaN}$ if u is a quiet NaN and x is not a signalling NaN

$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN or u is a signalling NaN

NOTE 2 – $arctanu_F(u, x) \approx arcu_F(u, 1, x)$.

5.3.7.13 Argument angular-unit arcus cotangentus operation

The $arccotu_F^*$ and $arcctgu_F^*$ approximation helper functions:

$arccotu_F^* : F \times F \to \mathcal{R}$

$arcctgu_F^* : F \times F \to \mathcal{R}$

$arccotu^*_F(u,x)$ returns a close approximation to $\text{arccot}(x) * u/(2*\pi)$ in $\mathcal{R}$, with maximum error $max\_error\_tanu_F(u)$.

$arcctgu^*_F(u,x)$ returns a close approximation to $\text{arcctg}(x) * u/(2*\pi)$ in $\mathcal{R}$, with maximum error $max\_error\_tanu_F(u)$.

There are two reasonable ways of selecting the principle value for the inverse of the cot oper. It is best to leave it to the user/programmer to decide which one is the most appropriate in a particular application. LIA-2 specifies both of them. Selecting just one is premature at the LIA level.

Further requirements on the $arccotu^*_F$ and $arccotu^*_F$ approximation helper functions:

$arccotu^*_F(u,1) = u/8$
$arccotu^*_F(u,0) = u/4$
$arccotu^*_F(u,-1) = 3*u/8$
$arccotu^*_F(u,x) \le u/2$ if $u > 0$
$arccotu^*_F(u,x) \ge u/2$ if $u < 0$
$arccotu^*_F(u,x) = u/2$ if $arccotu^*_F(u,x) \neq \text{arccot}(x) * u/(2*\pi)$ and $x < -3 * r_F/epsilon_F$
$arccotu^*_F(-u,x) = -arccotu^*_F(u,x)$
$arcctgu^*_F(u,x) = arccotu^*_F(u,x)$ if $x \ge 0$
$arcctgu^*_F(u,-x) = -arcctgu^*_F(u,x)$

The $arccotu_F$ operation:

$arccotu_F : F \times F \to F \cup \{\textbf{invalid}, \textbf{underflow}\}$

$arccotu_F(u,x) = trans\_result_F(arccotu^*_F(u,x))$ if $u \in G_F$ and $x \in F$
$= trans\_result_F(u/4)$ if $u \in G_F$ and $x = \textbf{-0}$
$= trans\_result_F(u/2)$ if $u \in G_F$ and $x = -\infty$
$= 0$ if $u \in G_F$ and $u > 0$ and $x = +\infty$
$= \textbf{-0}$ if $u \in G_F$ and $u < 0$ and $x = +\infty$

$= \textbf{invalid}(\textbf{qNaN})$ if $u \in F$ and $|u| < min\_angular\_unit_F$ and $x \in F \cup \{-\infty, \textbf{-0}, +\infty\}$
$= \textbf{invalid}(\textbf{qNaN})$ if $u \in \{-\infty, \textbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \textbf{-0}, +\infty\}$

$= \textbf{qNaN}$ if x is a quiet NaN and u is not a signalling NaN
$= \textbf{qNaN}$ if u is a quiet NaN and x is not a signalling NaN
$= \textbf{invalid}(\textbf{qNaN})$ if x is a signalling NaN or u is a signalling NaN

NOTE – $arccotu_F(u,x) \approx arcu_F(u,x,1)$.

The $arcctgu_F$ operation:

$arcctgu_F : F \times F \to F \cup \{\textbf{invalid}, \textbf{underflow}\}$

$arcctgu_F(u,x) = trans\_result_F(arcctgu^*_F(u,x))$ if $u \in G_F$ and $x \in F$
$= trans\_result_F(-u/4)$ if $u \in G_F$ and $x = \textbf{-0}$
$= \textbf{-0}$ if $u \in G_F$ and $u > 0$ and $x = -\infty$
$= 0$ if $u \in G_F$ and $u > 0$ and $x = +\infty$
$= 0$ if $u \in G_F$ and $u < 0$ and $x = -\infty$
$= \textbf{-0}$ if $u \in G_F$ and $u < 0$ and $x = +\infty$

$= \textbf{invalid}(\textbf{qNaN})$ if $u \in F$ and $|u| < min\_angular\_unit_F$ and

$x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{invalid(qNaN)}$ if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{qNaN}$ if x is a quiet NaN and u is not a signalling NaN
$= \mathbf{qNaN}$ if u is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid(qNaN)}$ if x is a signalling NaN or u is a signalling NaN

5.3.7.14 Argument angular-unit arcus secantus operation

The $arcsecu_F^*$ approximation helper function:

$arcsecu_F^* : F \times F \to \mathcal{R}$

$arcsecu_F^*(u, x)$ returns a close approximation to $\text{arcsec}(x) * u/(2 * \pi)$ in $\mathcal{R}$, with maximum error $max\_error\_tanu_F(u)$.

Further requirements on the $arcsecu_F^*$ approximation helper function:

$arcsecu_F^*(u, 2) = u/6$
$arcsecu_F^*(u, -2) = u/3$
$arcsecu_F^*(u, -1) = u/2$
$arcsecu_F^*(u, x) \leq u/4$ if $x > 0$ and $u > 0$
$arcsecu_F^*(u, x) \geq u/4$ if $x < 0$ and $u > 0$
$arcsecu_F^*(u, x) = u/4$ if $arcsecu_F^*(u, x) \neq \text{arcsec}(x) * u/(2 * \pi)$ and $|x| > 3 * r_F/epsilon_F$
$arcsecu_F^*(-u, x) = -arcsecu_F^*(u, x)$

The $arcsecu_F$ operation:

$arcsecu_F : F \times F \to F \cup \{\mathbf{underflow}, \mathbf{invalid}\}$

$arcsecu_F(u, x) = trans\_result_F(arcsecu_F^*(u, x))$ if $u \in G_F$ and $x \in F$ and $(x \leq -1$ or $x \geq 1)$

$= \mathbf{invalid(qNaN)}$ if $u \in G_F$ and $x \in F$ and $-1 < x < 1$
$= \mathbf{invalid(qNaN)}$ if $u \in G_F$ and $x = \mathbf{-0}$
$= trans\_result_F(u/4)$ if $u \in G_F$ and $x = -\infty$
$= trans\_result_F(u/4)$ if $u \in G_F$ and $x = +\infty$

$= \mathbf{invalid(qNaN)}$ if $u \in F$ and $|u| < min\_angular\_unit_F$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= \mathbf{invalid(qNaN)}$ if $u \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{qNaN}$ if x is a quiet NaN and u is not a signalling NaN
$= \mathbf{qNaN}$ if u is a quiet NaN and x is not a signalling NaN
$= \mathbf{invalid(qNaN)}$ if x is a signalling NaN or u is a signalling NaN

5.3.7.15 Argument angular-unit arcus cosecantus operation

The $arccscu_F^*$ approximation helper function:

$arccscu_F^* : F \times F \to \mathcal{R}$

$arccscu_F^*(u, x)$ returns a close approximation to $\text{arccsc}(x) * u/(2 * \pi)$ in $\mathcal{R}$, with maximum error $max\_error\_tanu_F(u)$.

Further requirements on the $arccscu_F^*$ approximation helper function:

$arccscu^*_F(u, 2) = u/12$
$arccscu^*_F(u, 1) = u/4$
$arccscu^*_F(u, -x) = -arccscu^*_F(u, x)$
$arccscu^*_F(-u, x) = -arccscu^*_F(u, x)$

The $arccscu_F$ operation:

$arccscu_F : F \times F \to F \cup \{\textbf{underflow}, \textbf{invalid}\}$

$arccscu_F(u, x) = trans\_result_F(arccscu^*_F(u, x))$

	if $u \in G_F$ and $x \in F$ and ($x \geq 1$ or $x \leq -1$)
$= \textbf{invalid}(\textbf{qNaN})$	if $u \in G_F$ and $x \in F$ and $-1 < x < 1$
$= \textbf{invalid}(\textbf{qNaN})$	if $u \in G_F$ and $x = \textbf{-0}$
$= \textbf{-0}$	if $u \in G_F$ and $u > 0$ and $x = -\infty$
$= 0$	if $u \in G_F$ and $u > 0$ and $x = +\infty$
$= 0$	if $u \in G_F$ and $u < 0$ and $x = -\infty$
$= \textbf{-0}$	if $u \in G_F$ and $u < 0$ and $x = +\infty$
$= \textbf{invalid}(\textbf{qNaN})$	if $u \in F$ and $u < min\_angular\_unit_F$ and $x \in F \cup \{-\infty, \textbf{-0}, +\infty\}$
$= \textbf{invalid}(\textbf{qNaN})$	if $u \in \{-\infty, \textbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \textbf{-0}, +\infty\}$
$= \textbf{qNaN}$	if x is a quiet NaN and u is not a signalling NaN
$= \textbf{qNaN}$	if u is a quiet NaN and x is not a signalling NaN
$= \textbf{invalid}(\textbf{qNaN})$	if x is a signalling NaN or u is a signalling NaN

5.3.8 Operations for degree trigonometrics and inverse degree trigonometrics

$deg_F : F \to F \cup \{\textbf{-0}, \textbf{angle\_too\_big}\}$
$deg_F(x) = unit_F(360, x)$

$sind_F : F \to F \cup \{\textbf{-0}, \textbf{underflow}, \textbf{angle\_too\_big}\}$
$sind_F(x) = sinu_F(360, x)$

$cosd_F : F \to F \cup \{\textbf{underflow}, \textbf{angle\_too\_big}\}$
$cosd_F(x) = cosu_F(360, x)$

$cossind_F : F \to F \times (F \cup \{\textbf{-0}\}) \cup \{\textbf{underflow}, \textbf{angle\_too\_big}\}$
$cossind_F(x) = cossinu_F(360, x)$

$tand_F : F \to F \cup \{\textbf{-0}, \textbf{pole}, \textbf{floating\_overflow}, \textbf{underflow}, \textbf{angle\_too\_big}\}$
$tand_F(x) = tanu_F(360, x)$

$cotd_F : F \to F \cup \{\textbf{pole}, \textbf{floating\_overflow}, \textbf{underflow}, \textbf{angle\_too\_big}\}$
$cotd_F(x) = cotu_F(360, x)$

$secd_F : F \to F \cup \{\textbf{pole}, \textbf{floating\_overflow}, \textbf{angle\_too\_big}\}$
$secd_F(x) = secu_F(360, x)$

$cscd_F : F \to F \cup \{\textbf{pole}, \textbf{floating\_overflow}, \textbf{angle\_too\_big}\}$
$cscd_F(x) = cscu_F(360, x)$

$arcsind_F : F \to F \cup \{\textbf{underflow}, \textbf{invalid}\}$
$arcsind_F(x) = arcsinu_F(360, x)$

$arccosd_F : F \to F \cup \{\textbf{invalid}\}$
$arccosd_F(x) = arccosu_F(360, x)$

$arcd_F : F \times F \to F \cup \{\textbf{underflow}, \textbf{invalid}\}$
$arcd_F(x, y) = arcu_F(360, x, y)$

$arctand_F : F \to F \cup \{\textbf{underflow}\}$
$arctand_F(x) = arctanu_F(360, x)$

$arccotd_F : F \to F \cup \{\textbf{underflow}\}$
$arccotd_F(x) = arccotu_F(360, x)$

$arcctgd_F : F \to F \cup \{\textbf{underflow}\}$
$arcctgd_F(x) = arcctgu_F(360, x)$

$arcsecd_F : F \to F \cup \{\textbf{invalid}\}$
$arcsecd_F(x) = arcsecu_F(360, x)$

$arccscd_F : F \to F \cup \{\textbf{underflow}, \textbf{invalid}\}$
$arccscd_F(x) = arccscu_F(360, x)$

5.3.9 Operations for angular-unit conversions

5.3.9.1 Converting radian angle to argument angular-unit angle

Define the mathematical function:

$rad\_to\_cycle : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$

$rad\_to\_cycle(x, v)$
$= \arccos(\cos(x)) * v/(2 * \pi)$ if $\sin(x) \geq 0$ and $v \neq 0$
$= -\arccos(\cos(x)) * v/(2 * \pi)$ if $\sin(x) < 0$ and $v \neq 0$

The $rad\_to\_cycle^*_F$ approximation helper function:

$rad\_to\_cycle^*_F : \mathcal{R} \times F \to \mathcal{R}$

$rad\_to\_cycle^*_F(x, v)$ returns a close approximation to $rad\_to\_cycle(x, v)$ in $\mathcal{R}$, with maximum error $max\_error\_rad_F$, if $|x| \leq big\_angle\_r_F$.

Further requirements on the $rad\_to\_cycle^*_F$ approximation helper function:

$rad\_to\_cycle^*_F(n * 2 * \pi + \pi/6, v) = v/12$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi/6| \leq big\_angle\_r_F$
$rad\_to\_cycle^*_F(n * 2 * \pi + \pi/4, v) = v/8$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi/4| \leq big\_angle\_r_F$
$rad\_to\_cycle^*_F(n * 2 * \pi + \pi/3, v) = v/6$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi/3| \leq big\_angle\_r_F$
$rad\_to\_cycle^*_F(n * 2 * \pi + \pi/2, v) = v/4$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi/2| \leq big\_angle\_r_F$
$rad\_to\_cycle^*_F(n * 2 * \pi + 2 * \pi/3, v) = v/3$
if $n \in \mathcal{Z}$ and $|n * 2 * \pi + 2 * \pi/3| \leq big\_angle\_r_F$
$rad\_to\_cycle^*_F(n * 2 * \pi + 3 * \pi/4, v) = 3 * v/8$
if $n \in \mathcal{Z}$ and $|n * 2 * \pi + 3 * \pi/4| \leq big\_angle\_r_F$
$rad\_to\_cycle^*_F(n * 2 * \pi + 5 * \pi/6, v) = 5 * v/12$
if $n \in \mathcal{Z}$ and $|n * 2 * \pi + 5 * \pi/6| \leq big\_angle\_r_F$
$rad\_to\_cycle^*_F(n * 2 * \pi + \pi, v) = v/2$ if $n \in \mathcal{Z}$ and $|n * 2 * \pi + \pi| \leq big\_angle\_r_F$
$rad\_to\_cycle^*_F(-x, v) = -rad\_to\_cycle^*_F(x, v)$
if $rad\_to\_cycle(x, v) \neq v/2$
$rad\_to\_cycle^*_F(x, -v) = -rad\_to\_cycle^*_F(x, v)$
if $rad\_to\_cycle(x, v) \neq v/2$

The $rad\_to\_cycle_F$ operation:

$rad\_to\_cycle_F : F \times F \to F \cup \{\textbf{underflow}, \textbf{angle\_too\_big}, \textbf{invalid}\}$

$rad\_to\_cycle_F(x, v)$

$= trans\_result_F(rad\_to\_cycle^*_F(x, v))$ if $v \in G_F$ and $x \in F$ and $|x| \le big\_angle\_r_F$

$= \mathbf{-0}$ if $v \in G_F$ and $x = \mathbf{-0}$

$= \mathbf{angle\_too\_big(qNaN)}$ if $v \in G_F$ and $x \in F$ and $|x| > big\_angle\_r_F$

$= \mathbf{invalid(qNaN)}$ if $v \in G_F$ and $x \in \{-\infty, +\infty\}$

$= \mathbf{invalid(qNaN)}$ if $v \in F$ and $|v| < min\_angular\_cycle_F$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{invalid(qNaN)}$ if $v \in \{-\infty, \mathbf{-0}, +\infty\}$ and $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$

$= \mathbf{qNaN}$ if x is a quiet NaN and v is not a signalling NaN

$= \mathbf{qNaN}$ if v is a quiet NaN and x is not a signalling NaN

$= \mathbf{invalid(qNaN)}$ if x is a signalling NaN or v is a signalling NaN

5.3.9.2 Converting argument angular-unit angle to radian angle

Define the mathematical function:

$cycle\_to\_rad : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$

$cycle\_to\_rad(u, x)$

$= \arccos(\cos(x * 2 * \pi/u))$ if $\sin(x * 2 * \pi/u) \ge 0$

$= -\arccos(\cos(x * 2 * \pi/u))$ if $\sin(x * 2 * \pi/u) < 0$

The $cycle\_to\_rad^*_F$ approximation helper function:

$cycle\_to\_rad^*_F : F \times \mathcal{R} \to \mathcal{R}$

$cycle\_to\_rad^*_F(u, x)$ returns a close approximation to $cycle\_to\_rad(u, x)$ in $\mathcal{R}$, if $u \ne 0$, with maximum error $max\_error\_rad_F$.

Further requirements on the $cycle\_to\_rad^*_F$ approximation helper function:

$cycle\_to\_rad^*_F(u, n * u + x) = cycle\_to\_rad^*_F(u, x)$ if $n \in \mathcal{Z}$

$cycle\_to\_rad^*_F(u, u/12) = \pi/6$

$cycle\_to\_rad^*_F(u, u/8) = \pi/4$

$cycle\_to\_rad^*_F(u, u/6) = \pi/3$

$cycle\_to\_rad^*_F(u, u/4) = \pi/2$

$cycle\_to\_rad^*_F(u, u/3) = 2 * \pi/3$

$cycle\_to\_rad^*_F(u, 3 * u/8) = 3 * \pi/4$

$cycle\_to\_rad^*_F(u, 5 * u/12) = 5 * \pi/6$

$cycle\_to\_rad^*_F(u, u/2) = \pi$

$cycle\_to\_rad^*_F(u, -x) = -cycle\_to\_rad^*_F(u, x)$ if $cycle\_to\_rad(u, x) \ne \pi$

The $cycle\_to\_rad_F$ operation:

$cycle\_to\_rad_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{angle\_too\_big}, \mathbf{invalid}\}$

$cycle\_to\_rad_F(u, x)$

$= trans\_result_F(cycle\_to\_rad^*_F(u, x))$ if $cycle_F(u, x) \in F$

$= \mathbf{-0}$ if $cycle_F(u, x) = \mathbf{-0}$

$= \mathbf{angle\_too\_big}$ if $cycle_F(u, x) = \mathbf{angle\_too\_big}$

$= \mathbf{invalid(qNaN)}$ if $cycle_F(u, x) = \mathbf{invalid}$

$= \mathbf{qNaN}$ if $cycle_F(u, x)$ is a quiet NaN

5.3.9.3 Converting argument angular-unit angle to (another) argument angular-unit angle

Define the mathematical function:

$$cycle\_to\_cycle : \mathcal{R} \times \mathcal{R} \times \mathcal{R} \to \mathcal{R}$$

$$\begin{aligned} cycle\_to\_cycle(u, x, v) \\ &= \arccos(\cos(x * 2 * \pi/u)) * v/(2 * \pi) \\ &\qquad \text{if } u \neq 0 \text{ and } v \neq 0 \text{ and } \sin(x * 2 * \pi/u) \geq 0 \\ &= -\arccos(\cos(x * 2 * \pi/u)) * v/(2 * \pi) \\ &\qquad \text{if } u \neq 0 \text{ and } v \neq 0 \text{ and } \sin(x * 2 * \pi/u) < 0 \end{aligned}$$

The $cycle\_to\_cycle^*_F$ approximation helper function:

$$cycle\_to\_cycle^*_F : F \times \mathcal{R} \times F \to \mathcal{R}$$

$cycle\_to\_cycle^*_F(u, x, v)$ returns a close approximation to $cycle\_to\_cycle(u, x, v)$ in $\mathcal{R}$ if $u \neq 0$ and $|x/u| \leq big\_angle\_u_F$, with maximum error $max\_error\_rad_F$.

Further requirements on the $cycle\_to\_cycle^*_F$ approximation helper function:

$$\begin{aligned} cycle\_to\_cycle^*_F(u, n * u + x, v) &= cycle\_to\_cycle^*_F(u, x, v) \\ &\qquad \text{if } n \in \mathcal{Z} \\ cycle\_to\_cycle^*_F(u, u/12, v) &= v/12 \\ cycle\_to\_cycle^*_F(u, u/8, v) &= v/8 \\ cycle\_to\_cycle^*_F(u, u/6, v) &= v/6 \\ cycle\_to\_cycle^*_F(u, u/4, v) &= v/4 \\ cycle\_to\_cycle^*_F(u, u/3, v) &= v/3 \\ cycle\_to\_cycle^*_F(u, 3 * u/8, v) &= 3 * v/8 \\ cycle\_to\_cycle^*_F(u, 5 * u/12, v) &= 5 * v/12 \\ cycle\_to\_cycle^*_F(u, u/2, v) &= v/2 \\ cycle\_to\_cycle^*_F(u, -x, v) &= -cycle\_to\_cycle^*_F(u, x, v) \\ &\qquad \text{if } cycle\_to\_cycle(u, x, v) \neq v/2 \\ cycle\_to\_cycle^*_F(-u, x, v) &= -cycle\_to\_cycle^*_F(u, x, v) \\ &\qquad \text{if } cycle\_to\_cycle(u, x, v) \neq v/2 \\ cycle\_to\_cycle^*_F(u, x, -v) &= -cycle\_to\_cycle^*_F(u, x, v) \\ &\qquad \text{if } cycle\_to\_cycle(u, x, v) \neq v/2 \end{aligned}$$

The $cycle\_to\_cycle_F$ operation:

$$cycle\_to\_cycle_F : F \times F \times F \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{angle\_too\_big}, \mathbf{invalid}\}$$

$$\begin{aligned} cycle\_to\_cycle_F(u, x, v) \\ &= trans\_result_F(cycle\_to\_cycle^*_F(u, x, v)) \\ &\qquad \text{if } v \in G_F \text{ and } cycle_F(u, x) \in F \\ &= \mathbf{-0} \qquad \text{if } v \in G_F \text{ and } cycle_F(u, x) = \mathbf{-0} \\ &= \mathbf{angle\_too\_big} \qquad \text{if } v \in G_F \text{ and } cycle_F(u, x) = \mathbf{angle\_too\_big} \\ &= \mathbf{invalid}(\mathbf{qNaN}) \qquad \text{if } v \in G_F \text{ and } cycle_F(u, x) = \mathbf{invalid} \\ &= \mathbf{invalid}(\mathbf{qNaN}) \qquad \text{if } v \in F \text{ and } v < min\_angular\_cycle_F \text{ and} \\ &\qquad\qquad u, x \in F \cup \{-\infty, \mathbf{-0}, +\infty\} \\ &= \mathbf{invalid}(\mathbf{qNaN}) \qquad \text{if } v \in \{-\infty, \mathbf{-0}, +\infty\} \text{ and } u, x \in F \cup \{-\infty, \mathbf{-0}, +\infty\} \\ &= \mathbf{qNaN} \qquad \text{if } cycle_F(u, x) \text{ is a quiet NaN and} \\ &\qquad\qquad v \text{ is not a signalling NaN} \\ &= \mathbf{qNaN} \qquad \text{if } v \text{ is a quiet NaN and} \\ &\qquad\qquad cycle_F(u, x) \neq \mathbf{invalid} \\ &= \mathbf{invalid}(\mathbf{qNaN}) \qquad \text{if } v \text{ is a signalling NaN} \end{aligned}$$

5.3.9.4 Degree angle conversions to and from other angular units

$rad\_to\_deg_F : F \to F \cup \{\mathbf{underflow}, \mathbf{angle\_too\_big}\}$
$rad\_to\_deg_F(x) = rad\_to\_cycle_F(x, 360)$

$deg\_to\_rad_F : F \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{angle\_too\_big}\}$
$deg\_to\_rad_F(x) = cycle\_to\_rad_F(360, x)$

$cycle\_to\_deg_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{angle\_too\_big}, \mathbf{invalid}\}$
$cycle\_to\_deg_F(u, x) = cycle\_to\_cycle_F(u, x, 360)$

$deg\_to\_cycle_F : F \times F \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{angle\_too\_big}, \mathbf{invalid}\}$
$deg\_to\_cycle_F(x, v) = cycle\_to\_cycle_F(360, x, v)$

5.4 Conversion operations

Fixed point string formats and floating point string formats should have formats for $\mathbf{-0}$, $+\infty$, $-\infty$, quiet and signalling NaNs. Integer string formats may have formats for such values.

NOTES

1 In ordinary string formats for numerals, the string "Hello world!" is an example of a signalling NaN.

2 This part of ISO/IEC 10967does not specify any string formats, not even for the special values $-\mathbf{0}$, $+\infty$, $-\infty$, and quiet NaN, but possibilities include the strings used in the text of this part of ISO/IEC 10967, as well as strings like "+infinity" or "positiva oändligheten", etc, and the strings used may depend on preference settings. String formats for numerical values, and if and how they may depend on preference settings, is also an issue for bindings or programming language specifications. It is not an issue for this part of ISO/IEC 10967.

5.4.1 Integer to integer conversions

I and I' represent different integer data types (even if the sets I and I' are the same). At least one of I and I' conform to LIA-1.

NOTE 1 – If both are I and I' are conforming to ISO/IEC 10967-1, then this conversion is covered by ISO/IEC 10967-1. This operation generalises the $cvt_{I \to I'}$ of ISO/IEC 10967-1:1994, with respect to that one of the integer types in the conversion need not be conforming to ISO/IEC 10967-1.

$convert_{I \to I'} : I \to I' \cup \{\mathbf{integer\_overflow}\}$

$$
\begin{array}{lll}
convert_{I \to I'}(x) = result_{I'}(x) & & \text{if } x \in I \\
= \mathbf{-0} & & \text{if } x = \mathbf{-0} \text{ and } \mathbf{-0} \text{ is available in the target type} \\
= -\infty & & \text{if } x = -\infty \text{ and } -\infty \text{ is available in the target type} \\
= +\infty & & \text{if } x = +\infty \text{ and } +\infty \text{ is available in the target type} \\
= 0 & & \text{if } x = \mathbf{-0} \text{ and } \mathbf{-0} \text{ is not available in the target type} \\
= \mathbf{integer\_overflow} & & \text{if } x = -\infty \text{ and } -\infty \text{ is not available in the target type} \\
= \mathbf{integer\_overflow} & & \text{if } x = +\infty \text{ and } +\infty \text{ is not available in the target type} \\
= \mathbf{qNaN} & & \text{if } x \text{ is a quiet NaN and quiet NaN is} \\
 & & \quad \text{available in the target type} \\
= \mathbf{invalid}(\mathbf{qNaN}) & & \text{if } x \text{ is a signalling NaN [}\mathbf{sNaN}\text{without notification?]}
\end{array}
$$

NOTE 2 – This covers, among other things, "input" and "output" of integer type values, including the possibility of overflow (for either input or output).

5.4.2 Floating point to integer conversions

I is an ISO/IEC 10967-1 conforming integer type. F is an ISO/IEC 10967-1 conforming floating point type.

NOTE – The operations in this clause are more specific than the floating point to integer conversion in ISO/IEC 10967-1:1994 which allows any rounding.

$rounding_{F\to I} : F \to I \cup \{\textbf{integer\_overflow}\}$

$rounding_{F\to I}(x)$

$= result_I(\text{round}(x))$	if $x \in F$
$= \textbf{-0}$	if $x = \textbf{-0}$ and $\textbf{-0}$ is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$	if $x = \textbf{-0}$ and $\textbf{-0}$ is not available in the target type
$= \textbf{integer\_overflow}$	if $x = -\infty$ and $-\infty$ is not available in the target type
$= \textbf{integer\_overflow}$	if $x = +\infty$ and $+\infty$ is not available in the target type
$= \textbf{qNaN}$	if x is a quiet NaN and quiet NaN is available in the target type
$= \textbf{invalid}(\textbf{qNaN})$	if x is a signalling NaN [**sNaN**without notification?]

$floor_{F\to I} : F \to I \cup \{\textbf{integer\_overflow}\}$

$floor_{F\to I}(x)$

$= result_I(\lfloor x \rfloor)$	if $x \in F$
$= \textbf{-0}$	if $x = \textbf{-0}$ and $\textbf{-0}$ is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$	if $x = \textbf{-0}$ and $\textbf{-0}$ is not available in the target type
$= \textbf{integer\_overflow}$	if $x = -\infty$ and $-\infty$ is not available in the target type
$= \textbf{integer\_overflow}$	if $x = +\infty$ and $+\infty$ is not available in the target type
$= \textbf{qNaN}$	if x is a quiet NaN and quiet NaN is available in the target type
$= \textbf{invalid}(\textbf{qNaN})$	if x is a signalling NaN [**sNaN**without notification?]

$ceiling_{F\to I} : F \to I \cup \{\textbf{integer\_overflow}\}$

$ceiling_{F\to I}(x)$

$= result_I(\lceil x \rceil)$	if $x \in F$
$= \textbf{-0}$	if $x = \textbf{-0}$ and $\textbf{-0}$ is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$	if $x = \textbf{-0}$ and $\textbf{-0}$ is not available in the target type
$= \textbf{integer\_overflow}$	if $x = -\infty$ and $-\infty$ is not available in the target type
$= \textbf{integer\_overflow}$	if $x = +\infty$ and $+\infty$ is not available in the target type
$= \textbf{qNaN}$	if x is a quiet NaN and quiet NaN is available in the target type
$= \textbf{invalid}(\textbf{qNaN})$	if x is a signalling NaN [**sNaN**without notification?]

5.4.3 Integer to floating point conversions

$convert^{nearest}_{I\to F} : I \to F \cup \{\textbf{floating\_overflow}\}$

$convert^{nearest}_{I\to F}(x)$

$= result_F(x, nearest_F)$	if $x \in I$
$= \textbf{-0}$	if $x = \textbf{-0}$ and $\textbf{-0}$ is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type

	$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
	$= 0$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is not available in the target type
	$=$ **floating_overflow**	if $x = -\infty$ and $-\infty$ is not available in the target type
	$=$ **floating_overflow**	if $x = +\infty$ and $+\infty$ is not available in the target type
	$=$ **qNaN**	if x is a quiet NaN and quiet NaN is available in the target type
	$=$ **invalid**(**qNaN**)	if x is a signalling NaN [**sNaN**without notification?]

The following two operations are to support interval arithmetic.

$convert^{down}_{I \to F} : I \to F \cup \{$**floating_overflow**$\}$

$convert^{down}_{I \to F}(x)$	$= result_F(x, down_F)$	if $x \in I$
	$= \mathbf{-0}$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is available in the target type
	$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
	$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
	$= 0$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is not available in the target type
	$=$ **floating_overflow**	if $x = -\infty$ and $-\infty$ is not available in the target type
	$=$ **floating_overflow**	if $x = +\infty$ and $+\infty$ is not available in the target type
	$=$ **qNaN**	if x is a quiet NaN and quiet NaN is available in the target type
	$=$ **invalid**(**qNaN**)	if x is a signalling NaN [**sNaN**without notification?]

$convert^{up}_{I \to F} : I \to F \cup \{$**floating_overflow**$\}$

$convert^{up}_{I \to F}(x)$	$= result_F(x, up_F)$	if $x \in I$
	$= \mathbf{-0}$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is available in the target type
	$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
	$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
	$= 0$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is not available in the target type
	$=$ **floating_overflow**	if $x = -\infty$ and $-\infty$ is not available in the target type
	$=$ **floating_overflow**	if $x = +\infty$ and $+\infty$ is not available in the target type
	$=$ **qNaN**	if x is a quiet NaN and quiet NaN is available in the target type
	$=$ **invalid**(**qNaN**)	if x is a signalling NaN [**sNaN**without notification?]

NOTE – Integer to nearest floating point conversions are covered by ISO/IEC 10967-1. I.e. $cvt_{I \to F} = convert^{nearest}_{I \to F}$, when both I and F conform to LIA-1.

5.4.4 Floating point to floating point conversions

F and F' represent different floating point data types (even if the sets F and F' are the same). At least one of F and F' conform to LIA-1.

$convert^{nearest}_{F \to F'} : F \to F' \cup \{$**floating_overflow**, **underflow**$\}$

$convert^{nearest}_{F \to F'}(x)$		
	$= result_{F'}(x, nearest_{F'})$	if $x \in F$ [gen. $cvt_{F \to F'}$ of LIA-1]
	$= \mathbf{-0}$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is available in the target type
	$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
	$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
	$= 0$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is not available in the target type
	$=$ **floating_overflow**	if $x = -\infty$ and $-\infty$ is not available in the target type
	$=$ **floating_overflow**	if $x = +\infty$ and $+\infty$ is not available in the target type
	$=$ **qNaN**	if x is a quiet NaN and quiet NaN is

available in the target type
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN [**sNaN**without notification?]

The following two operations are to support interval arithmetic.

$convert^{down}_{F \to F'} : F \to F' \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$

$convert^{down}_{F \to F'}(x)$

$= result_{F'}(x, down_{F'})$ if $x \in F$
$= \mathbf{-0}$ if $x = \mathbf{-0}$ and $\mathbf{-0}$ is available in the target type
$= -\infty$ if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$ if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$ if $x = \mathbf{-0}$ and $\mathbf{-0}$ is not available in the target type
$= \mathbf{floating\_overflow}$ if $x = -\infty$ and $-\infty$ is not available in the target type
$= \mathbf{floating\_overflow}$ if $x = +\infty$ and $+\infty$ is not available in the target type
$= \mathbf{qNaN}$ if x is a quiet NaN and quiet NaN is available in the target type
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN [**sNaN**without notification?]

$convert^{up}_{F \to F'} : F \to F' \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$

$convert^{up}_{F \to F'}(x)$

$= result_{F'}(x, up_{F'})$ if $x \in F$
$= \mathbf{-0}$ if $x = \mathbf{-0}$ and $\mathbf{-0}$ is available in the target type
$= -\infty$ if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$ if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$ if $x = \mathbf{-0}$ and $\mathbf{-0}$ is not available in the target type
$= \mathbf{floating\_overflow}$ if $x = -\infty$ and $-\infty$ is not available in the target type
$= \mathbf{floating\_overflow}$ if $x = +\infty$ and $+\infty$ is not available in the target type
$= \mathbf{qNaN}$ if x is a quiet NaN and quiet NaN is available in the target type
$= \mathbf{invalid}(\mathbf{qNaN})$ if x is a signalling NaN [**sNaN**without notification?]

NOTES

1 Floating point to nearest floating point conversions are covered by ISO/IEC 10967-1 when both types conform to ISO/IEC 10967-1.

2 This covers, among other things, "input" and "output" of floating point type values, for floating point string formats.

5.4.5 Floating point to fixed point conversions

D is a fixed point type (essentially LID scaled, but it may be limited). A fixed point type D shall be a subset of $\mathcal{R}$, characterised by a radix, $r_D \in \mathcal{Z}$ (≥ 2), a density, $d_D \in \mathcal{Z}$ (≥ 0), and if it is limited a maximum positive value, $dmax_D \in D^*$ (≥ 1). Given these values, the following sets are defined:

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

$D = D^*$ if D is unlimited
$= D^* \cap [-dmax_D, dmax_D]$ if D is limited

Fixed point rounding helper functions:

$down_D : \mathcal{R} \to D^*$ is a rounding function that rounds towards negative infinity.

$up_D : \mathcal{R} \to D^*$ is a rounding function that rounds towards positive infinity.

$nearest_D : \mathcal{R} \to D^*$ is a rounding function that rounds to nearest, ties round to even last digit.

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

$result_D : \mathcal{R} \times (\mathcal{R} \to D^*) \to D \cup \{\textbf{overflow}\}$

$result_D(x, rnd)$	$= rnd(x)$	if $rnd(x) \in D$ and ($rnd(x) \neq 0$ or $x \geq 0$)
	$= \textbf{−0}$	if $rnd(x) = 0$ and $x < 0$ and **−0** available
	$= 0$	if $rnd(x) = 0$ and $x < 0$ and **−0** not available
	$= \textbf{overflow}$	if $x in \mathcal{R}$ and $rnd(x) \notin D$

F is a floating point type conforming to LIA-1. D is a fixed point type, it cannot conform to LIA-1, since fixed point types are not covered by LIA-1.

$convert^{nearest}_{F \to D} : F \to D \cup \{\textbf{−0}, \textbf{overflow}\}$

$convert^{nearest}_{F \to D}(x)$

$= resultD(x, nearestD)$	if $x \in F$
$= \textbf{−0}$	if $x =$ **−0** and **−0** is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$	if $x =$ **−0** and **−0** is not available in the target type
$= \textbf{overflow}$	if $x = -\infty$ and $-\infty$ is not available in the target type
$= \textbf{overflow}$	if $x = +\infty$ and $+\infty$ is not available in the target type
$= \textbf{qNaN}$	if x is a quiet NaN and quiet NaN is available in the target type
$= \textbf{invalid}(\textbf{qNaN})$	if x is a signalling NaN [**sNaN**without notification?]

$convert^{down}_{F \to D} : F \to D \cup \{\textbf{overflow}\}$

$convert^{down}_{F \to D}(x)$

$= result_D(x, down_D)$	if $x \in F$
$= \textbf{−0}$	if $x =$ **−0** and **−0** is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$	if $x =$ **−0** and **−0** is not available in the target type
$= \textbf{overflow}$	if $x = -\infty$ and $-\infty$ is not available in the target type
$= \textbf{overflow}$	if $x = +\infty$ and $+\infty$ is not available in the target type
$= \textbf{qNaN}$	if x is a quiet NaN and quiet NaN is available in the target type
$= \textbf{invalid}(\textbf{qNaN})$	if x is a signalling NaN [**sNaN**without notification?]

$convert^{up}_{F \to D} : F \to D \cup \{\textbf{−0}, \textbf{overflow}\}$

$convert^{up}_{F \to D}(x)$

$= result_D(x, up_D)$	if $x \in F$
$= \textbf{−0}$	if $x =$ **−0** and **−0** is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$	if $x =$ **−0** and **−0** is not available in the target type
$= \textbf{overflow}$	if $x = -\infty$ and $-\infty$ is not available in the target type
$= \textbf{overflow}$	if $x = +\infty$ and $+\infty$ is not available in the target type
$= \textbf{qNaN}$	if x is a quiet NaN and quiet NaN is available in the target type

$= \mathbf{invalid(qNaN)}$ if x is a signalling NaN [**sNaN**without notification?]

NOTES

1 The type D need not be visible in the programming language. D may be a subtype of strings, according to some format. Even so, no type for strings need be present in the programming language.

2 This covers, among other things, "output" of floating point type values, to fixed point string formats. E.g. a binding may say that `float_to_fixed_string(`x`, `m`, `n`)` is bound to $convertnearest_F(Sm, n(x)$ where $S_{m,n}$ is strings of length m, representing fixed point values in (LID) **scaled(10, n)**. The binding should also detail how NaNs, signed zeroes and infinities are represented in $S_{m,n}$, as well as the precise format of the strings representing ordinary values. (Note that if the length of the target string is limited, the conversion may overflow.)

5.4.6 Fixed point to floating point conversions

F is a floating point type conforming to ISO/IEC 10967-1. D is a fixed point type, it cannot conform to ISO/IEC 10967-1, since fixed point types are not covered by ISO/IEC 10967-1.

$convert^{nearest}_{D\to F} : D \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$

$convert^{nearest}_{D\to F}(x)$

$= result_F(x, nearest_F)$	if $x \in D$
$= \mathbf{-0}$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is not available in the target type
$= \mathbf{overflow}$	if $x = -\infty$ and $-\infty$ is not available in the target type
$= \mathbf{overflow}$	if $x = +\infty$ and $+\infty$ is not available in the target type
$= \mathbf{qNaN}$	if x is a quiet NaN and quiet NaN is available in the target type
$= \mathbf{invalid(qNaN)}$	if x is a signalling NaN [**sNaN**without notification?]

$convert^{down}_{D\to F} : D \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$

$convert^{down}_{D\to F}(x)$

$= result_F(x, down_F)$	if $x \in D$
$= \mathbf{-0}$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type
$= 0$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is not available in the target type
$= \mathbf{overflow}$	if $x = -\infty$ and $-\infty$ is not available in the target type
$= \mathbf{overflow}$	if $x = +\infty$ and $+\infty$ is not available in the target type
$= \mathbf{qNaN}$	if x is a quiet NaN and quiet NaN is available in the target type
$= \mathbf{invalid(qNaN)}$	if x is a signalling NaN [**sNaN** without notification?]

$convert^{up}_{D\to F} : D \to F \cup \{\mathbf{floating\_overflow}, \mathbf{underflow}\}$

$convert^{up}_{D\to F}(x)$

$= result_F(x, up_F)$	if $x \in D$
$= \mathbf{-0}$	if $x = \mathbf{-0}$ and $\mathbf{-0}$ is available in the target type
$= -\infty$	if $x = -\infty$ and $-\infty$ is available in the target type
$= +\infty$	if $x = +\infty$ and $+\infty$ is available in the target type

$= 0$	if $x = -\mathbf{0}$ and $-\mathbf{0}$ is not available in the target type
$=$ **overflow**	if $x = -\infty$ and $-\infty$ is not available in the target type
$=$ **overflow**	if $x = +\infty$ and $+\infty$ is not available in the target type
$=$ **qNaN**	if x is a quiet NaN and quiet NaN is available in the target type
$=$ **invalid**(**qNaN**)	if x is a signalling NaN [**sNaN** without notification?]

NOTE – This covers, among other things, "input" of floating point type values, from fixed point string formats. E.g. a binding may say that `string_to_float(`s`)` is bound to $convert^{nearest}_{S_{m,n} \to F}(s)$ where $S_{m,n}$ is strings of length m, where m is the length of s, and n is the number of digits after the "decimal symbol" in s. The binding should also detail how NaNs, signed zeroes and infinities are represented in $S_{m,n}$, as well as the precise format of the strings representing ordinary values.

5.5 Numerals

Each numeral is an operation. Thus, this clause introduces a very large number of operations, since the number of numerals is in principle infinite.

5.5.1 Numerals for integer types

A numeral, denoting a mathematical value n in $\mathcal{Z}$, for an integer type, I, results in

$result_I(n)$

For each ISO/IEC 10967-1 conforming integer type there shall be integer numerals for all non-negative values of I. Integer numeral representations using radix 10 should be available.

NOTES

1 Negative values (except $minint_I$ if $minint_I = -maxint_I - 1$) can be obtained by using the negation operation (neg_I).

2 Other radices may also be available for integer numerals, and the radix used may be part of determining the goal integer type. E.g., radix 10 may be for signed integer types, and radix 8 or 16 may be for unsigned integer types.

3 Syntaxes for numerals for different integer types need not be different, nor need they be the same. LIA-2 does not further specify the format for integer numerals. That is an issue for bindings.

4 Overflow for numerals can be detected at "compile time", and warned about.

5.5.2 Numerals for floating point types

A fractional numeral, denoting a mathematical value x in $\mathcal{R}$, for a floating point type, F, shall normally result in:

$result_F(x, nearest_F)$

shall in a round towards negative infinity circumstance result in:

$result_F(x, down_F)$

shall in a round towards positive infinity circumstance result in:

$result_F(x, up_F)$

If iec_559_F = **true** then the directed roundings shall be available also for floating point numerals. The rounding circumstance should be statically determined, if other than the normal is at all available.

For each ISO/IEC 10967-1 conforming floating point type, F, there shall be fractional numerals for all radix 10 limited precision and limited range expressible non-negative values of $\mathcal{R}$. The precision and range for the numerals shall be large enough to allow all non-negative values of F to be reachable.

There shall be a numeral for positive infinity. There shall be numerals for quiet and signalling NaNs.

NOTES

1 Negative values (including negative 0, $-\mathbf{0}$) can be obtained by using the negation operation (neg_F).

2 Other radices may also be available for floating point numerals.

3 Integer numerals may also be fractional numerals, i.e. their syntaxes need not be different. Nor need syntaxes for numerals for different floating point types be different, nor need they be the same. ISO/IEC 10967-2 does not specify the syntax for numerals. That is an issue for bindings or programming language specifications.

6 Notification

Notification is the process by which a user or program is informed that a arithmetic operation cannot be performed so that a result within the error bounds is returned. Specifically, a notification shall occur when any such operation returns one of the exceptional values: **integer_overflow**, **undefined**, **invalid**, **pole**, **underflow**, **floating_overflow**, and **angle_too_big**. The exceptional value involved is called the *kind* of notification that occurs.

Notification shall be performed according to the requirements of clause 6 of ISO/IEC 10967-1. If notifications are handled by a recording of indicators (see clause 6.1.2 of ISO/IEC 10967-1), the implementation shall provide (and document) a *continuation value* for the result of the failed operation, if that value differs from what is specified in ISO/IEC 10967-2.

An implementation shall suppress spurious notifications.

NOTE 1 – E.g., an intermediate overflow on computing approximations to x^2 or y^2 during the calculation of $hypot_F(x, y) \approx \sqrt{x^2 + y^2}$). This is clear from the ISO/IEC 10967-2 specification of the $hypot_F$ operation.

2 If an operation op_F, for the corresponding mathematical function f, is such that $f(x)$ very closely approximates x, when $|x| \leq fminN_F$, then $op_F(x)$ returns x for $|x| \leq fminN_F$, and does not signal an exception if there is no denormalisation loss. For details, see the individual operation specifications for $expm1_F$, $ln1p_F$, $sinh_F$, $arcsinh_F$, $tanh_F$, $arctanh_F$, sin_F, $arcsin_F$, tan_F, and $arctan_F$.

Floating point datatypes that satisfy the requirements of IEC 559 have special values in addition to the values in F. These are: $-\mathbf{0}$, $+\infty$, $-\infty$, *signaling* **NaN**s (**sNaN**), and *quiet* **NaN**s (**qNaN**). Such values may be passed as arguments to operations, and used as results or continuation values. Floating point types that do not fully conform to IEC 559 might also have values corresponding to $-\mathbf{0}$, $+\infty$, $-\infty$, or **NaN**.

Most operations specified in ISO/IEC 10967-2 return **invalid**(**qNaN**) when passed a signaling **NaN** (**sNaN**) as an argument. Most operations specified in ISO/IEC 10967-2 return **qNaN**, without any notification when passed a quiet **NaN** (**qNaN**) as an argument.

The results of passing special values to operations is found in the operation specifications.

NOTES

3 The different kinds of notifications occur under the following circumstances:

a) **invalid**: when an argument is not valid for the operation, and no value in F^* or any special value result makes mathematical sense.

b) **pole**: when the input operand corresponds to a pole of the mathematical function approximated by the operation.

c) **integer_overflow**: when the (integer) result is outside of the range of the result datatype.

d) **floating_overflow**: when a sufficiently closely approximating result of the operation has a magnitude that is too large to be accurately represented in the result datatype.

e) **underflow**: when a sufficiently closely approximating result of the operation has a magnitude that is too small to be accurately represented in the result datatype.

f) **angle_too_big**: when the magnitude of the angle argument of a trigonometric operation exceeds the maximum value of the argument for which the density of floating point values is deemed sufficient for the operation to make sense. See clause 5.3.5 and the detailed discussion in clause A.5.3.5.

4 See A.6 for a discussion of the omission of an **underflow** notification under the circumstances mentioned above.

5 The difference between the **pole** and **floating_overflow** notifications is that the first corresponds to a true mathematical singularity, and the second corresponds to a well-defined mathematical result that happens to lie outside the range of F.

6 Signalling **NaN**s are not produced by any operation in ISO/IEC 10967-2.

6.1 Continuation values

Continuation values of $-\mathbf{0}$, $+\infty$, $-\infty$, and **NaN** are required only if the parameter iec_559_F has the value **true**. If the implementation can represent such special values in the result datatype, they should be used according to the specifications in ISO/IEC 10967-2. The distinction between signaling and quiet **NaN**s is required only if the implementation is capable of making such a distinction.

When the notification process requires a continuation value, the following requirements (organized by notification kind) shall be satisfied for operations with floating point result.

For a **invalid** notification, the continuation value shall be a quiet **NaN**, unless specified explicitly otherwise.

If there are quiet **NaN**s among the arguments, a quiet **NaN** shall be used as the continuation value, unless specified explicitly otherwise.

For a **floating_overflow** notification, the continuation value shall be as given in parentheses following the exception value in the specification.

For a **pole** notification, the continuation value shall be as given in parentheses following the exception value in the specification.

For an **underflow** notification, the continuation value shall be one of $fminN$, $-fminN$, or a subnormal value.

For a **angle_too_big** notification, the continuation value shall be a **NaN**.

NOTES

1 The prescribed continuation values for **floating_overflow** and **pole** are $+\infty$ or $-\infty$.

2 In order to avoid **angle_too_big** notifications, and to maintain a high accuracy, implementors are encouraged to provide, and programmers encouraged to use, the angle normalisation operations specified in 5.3.6.1, 5.3.7.1, and 5.3.8.

7 Relationship with language standards

A computing system often provides some of the operations specified in ISO/IEC 10967-2 within the context of a standard programming language. The requirements of the present standard shall be in addition to those imposed by the relevant programming language standards.

This standard does not define the syntax of arithmetic expressions. However, programmers need to know how to reliably access the operations defined in this standard.

NOTE 1 – Providing the information required in this clause is properly the responsibility of programming language standards. An individual implementation would only need to provide details if it could not cite an appropriate clause of the language or binding standard.

An implementation shall document the notation used to invoke each operation specified in this part of ISO/IEC 10967.

NOTE 2 – For example, the radian arc sine operation ($arcsin_F(x)$) might be invoked as

`arcsin(x)`	in Pascal [5] and Ada [6]
`asin(x)`	in C [9] and Fortran [3]
`(asin x)`	in Common Lisp and ISLisp

An implementation shall document the semantics of arithmetic expressions in terms of compositions of the operations specified in clause 5 of this part of ISO/IEC 10967and of clause 5 of ISO/IEC 10967-1.

NOTE 3 – For example, if x is declared to be single precision (`SP`) floating point, and calculation is done in single precision, then the expression

`arcsin(x)`

might translate to

$arcsin_{SP}$(`x`)

If the language in question did all computations in double precision (`DP`) floating point, the above expression might translate to

$cvt_{DP \to SP}(arcsin_{DP}(cvt_{SP \to DP}(x)))$

Alternatively, if `x` was declared to be an integer, and the expected result datatype is single precision float, the above expression might translate to

$cvt_{DP \to SP}(arcsin_{DP}(cvt_{I \to DP}(x)))$

Compilers often "optimize" code as part of compilation. Thus, an arithmetic expression might not be executed as written. An implementation shall document the possible transformations of arithmetic expressions (or groups of expressions) that it permits. Typical transformations include

a) Insertion of operations, such as data type conversions or changes in precision.

b) Replacing operations (or entire subexpressions) with others, such as "`cos(-x)`" → "`cos(x)`" (exactly the same result) or "`pi - arccos(x)`" → "`arccos(-x)`" (more accurate result) or "`exp(x)-1`" → "`expm1(x)`" (more accurate result if $x > -1$, less accurate result if $x < -1$, different notification behaviour).

c) Evaluating constant subexpressions.

d) Eliminating unneeded subexpressions.

Only transformations which alter the semantics of an expression (the values produced, and the notifications generated) need be documented. Only the range of permitted transformations need be documented. It is not necessary to describe the specific choice of transformations that will be applied to a particular expression. (See the Fortran standard [3], particularly clauses 7.1.2 and 7.1.7, for an example of documentation in this area.)

The textual scope of such transformations shall be documented, and any mechanisms that provide programmer control over this process should be documented as well.

NOTE 4 – It is highly desirable that programming languages intended for numerical use provide means for limiting the transformations applied to particular arithmetic expressions. Control over changes of precision is particularly useful.

8 Documentation requirements

In order to conform to ISO/IEC 10967-2, an implementation shall include documentation providing the following information to programmers.

NOTE 1 – Much of the documentation required in this clause is properly the responsibility of programming language or binding standards. An individual implementation would only need to provide details if it could not cite an appropriate clause of the language or binding standard.

a) A list of the provided operations that conform to ISO/IEC 10967-2.

b) For each maximum error parameter, the value of that parameter. Only parameters that are relevant to the provided operations need be given.

c) The value of the parameter $big\_angle\_r_F$ and the definition of the parameter function $big\_angle\_u_F$.

d) For the $nearest_F$ function, the rule used for rounding halfway cases.

e) For each conforming operation, the continuation value provided for each notification condition. Specific continuation values that are required by ISO/IEC 10967-2 need not be documented. If the notification mechanism does not make use of continuation values (see clause 6), continuation values need not be documented.

NOTE 2 – Implementations that do not provide infinities or **NaN**s will have to document any continuation values used in place of such values.

f) For each conforming operation, how the results depend on the rounding mode, if rounding modes are provided. Operations may be insensitive to the rounding mode, or sensitive to it, but even then need not heed the rounding mode.

g) For each conforming operation, the notation to be used for invoking that operation.

h) For each maximum error parameter, the notation to be used to access that parameter.

i) The notation to be used to access the parameters $big\_angle\_r_F$ and $big\_angle\_u_F(u)$.

Since the integer and floating point types used in conforming operations shall satisfy the requirements of ISO/IEC 10967-1, the following information shall also be provided by any conforming implementation.

j) The translation of arithmetic expressions into combinations of the operations provided by any part of ISO/IEC 10967, including any use made of higher precision. (See clause 7 of ISO/IEC 10967-1.)

k) The methods used for notification, and the information made available about the violation. (See clause 6 of ISO/IEC 10967-1.)

l) The means for selecting among the notification methods, and the notification method used in the absence of a user selection. (See 6.3 of ISO/IEC 10967-1.)

m) The means for selecting the modes of operation that ensure conformity.

n) When "recording of indicators" is the method of notification, the datatype used to represent *Ind*, the method for denoting the values of *Ind* (the association of these values with the subsets of E must be clear), and the notation for invoking each of the "indicator" operations. (See 6.1.2 of ISO/IEC 10967-1.)

In interpreting 6.1.2 of ISO/IEC 10967-1, the set of indicators E shall be interpreted as including all exceptional values listed in the signatures of conforming operations. In particular, E need to contain **pole** and **angle_too_big**.

Annex A
(informative)

Rationale

This annex explains and clarifies some of the ideas behind *Information technology – Language independent arithmetic – Part 2: Elementary numerical functions* (LIA-2). This allows the standard itself to be concise.

A.1 Scope

A.1.1 Specifications included in ISO/IEC 10967-2

This part of ISO/IEC 10967(LIA-2) is intened to define the meaning of some additional operations on integer and floating point types as specified in ISO/IEC 10967-1. ISO/IEC 10967-2 does not specify any additional arithmetic datatypes, though fixed point datatypes are used in some of the specifications for conversion operations.

The specifications for the operations covered by ISO/IEC 10967-2 are given in Sufficient detail to

a) support detailed and accurate numerical analysis of arithmetic algorithms, enable a precise determination of conformity or non-conformity,

b) prevent exceptions (like overflow) from going undetected.

A.1.2 Specifications not within the scope of ISO/IEC 10967-2

ISO/IEC 10967-2 is not concerned with techniques for the implementation of portable numerical functions.

ISO/IEC 10967-2 does not provide specifications for operations which involve no arithmetic processing. It also omits operations for the support of specialised mathematical domains such as linear algebra, statistics, and symbolic processing. Such domains deserve separate standardisation.

A.2 Conformity

Conformanity to this standard is dependent on the existence of language binding standards. Each language committee is encouraged to produce a binding standard covering at least those operations already required by the language standard and also specified in ISO/IEC 10967-2.

The term “language standard” in the previous paragraph is used in a generalised sense to include other computing entities such as calculators, spread sheets, and database query languages to the extent that they provide the operations covered in ISO/IEC 10967-2.

Suggestions for bindings are provided in Annex C. Annex C has partial binding examples for a number of existing languages and ISO/IEC 10967-2.

In addition to the bindings for the operations in ISO/IEC 10967-2, it is also necessary to provide bindings for the maximum error parameters and big angle parameters. Annex C contains suggestions for these bindings.

To conform to this standard, in the absence of a binding standard, an implementation should create a binding, following the suggestions in Annex C.

A.3 Normative references

A.4 Symbols and definitions

A.4.1 Symbols

The sequence types $[I]$ and $[F]$ appear as input to a few operations. In effect, a sequence is a finite linearly ordered collection of elements which can be indexed from 1 to the length of the sequence. Equality of two or more elements with different indices is possible.

A helper function from ISO/IEC 10967-1 is used in the conversion of input data into internal form. This function, $result_F$, is defined in clause 5.2.6 of ISO/IEC 10967-1, has the following signature:

$$result_F : \mathcal{R} \times (\mathcal{R} \to F^*) \to F \cup \{\textbf{floating\_overflow}, \textbf{underflow}\}$$

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

For all values $x \in \mathcal{R}$, and any rounding function rnd in $(\mathcal{R} \to F^*)$, the following shall apply:

For $x = 0$ or $fminN \le |x| \le fmax$:

$$result_F(x, rnd) = rnd(x)$$

For $|x| > fmax$:

$$\begin{aligned} result_F(x, rnd) &= rnd(x) && \text{if } |rnd(x)| = fmax \\ &= \textbf{floating\_overflow} && \text{otherwise} \end{aligned}$$

For $0 < |x| < fminN$:

$$\begin{aligned} result_F(x, rnd) &= rnd(x) \text{ or } \textbf{underflow} && \text{if } |rnd(x)| = fminN \\ &= rnd(x) \text{ or } \textbf{underflow} && \text{if } |rnd(x)| \in F_D,\ denorm = \textbf{true}, \text{ and} \\ & && \quad rnd \text{ has no denormalization loss at } x \\ &= \textbf{underflow} && \text{otherwise} \end{aligned}$$

An implementation is allowed to choose between $rnd(x)$ and **underflow** in the region between 0 and $fminN$. However, a denormalised value for $rnd(x)$ can be chosen only if $denorm$ is **true** and no denormalisation loss occurs at x. An implementation shall document how the choice between $rnd(x)$ and **underflow** is made.

A second helper function $wrap_I$ produces x if $x \in I$ and a wrapped result otherwise. The definition in clause 5.1.2 of ISO/IEC 10967-1:1994 is

$$wrap_I : \mathcal{Z} \to I$$
$$wrap_I(x) = x + j * (maxint - minint + 1) \qquad \text{for some } j \in \mathcal{Z}$$

A.4.2 Definitions

A.5 Specifications for the numerical functions

A.5.1 Additional basic integer operations

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

The $result_I$ helper function notifies overflow when the result cannot be represented in I.

The $wrap_I$ helper function wraps the result into a value that can be represented in I. The result is wrapped in such a way that the value returned can be used in extended range integer arithmetic.

A.5.1.2 Integer maximum and minimum operations

A.5.1.3 Integer positive difference (monus, diminish) operation

A.5.1.4 Integer power and arithmetic shift operations

The integer arithmetic shift operations can be used to implement integer multiplication and integer division more quickly in special cases.

A.5.1.5 Integer square root (rounded to nearest integer) operation

A.5.1.6 Divisibility and even/odd test operations

A.5.1.7 Greatest common divisor and least common multiple operations

The greatest common divisor is useful in reducing a fraction (a rational number) to its lowest terms, without loosing accuracy.

The least common multiple is useful in converting two fractions (rational numbers) to have the same denominator.

A.5.1.8 Support operations for extended integer range

These operations would typically be used to extend the range of the highest level supported by the underlying hardware of an implementation.

The two parts of an integer product, $mul\_ov_I(x, y)$ and $mul\_wrap_I(x, y)$ together provide the complete integer product. Similarly for addition and subtraction.

The use of $wrap_I$ guarantees that **integer_overflow** will not occur.

A.5.2 Additional basic floating point operations

A.5.2.1 The rounding and floating point *result* helper functions

A.5.2.2 Floating point maximum and minimum operations

A.5.2.3 Floating point positive difference (monus, diminish) operation

A.5.2.4 Round, floor, and ceiling operations

Since $fmax_F$ always has an integral value according to ISO/IEC 10967-1, no overflow can occur for these operations.

A.5.2.5 Operation for remainder after division and round to integer (IEEE remainder)

The remainder after division and round to integer (IEC 559 remainder) is an exact operation, even if the floating point datatype only conforms to ISO/IEC 10967-1, but not to the more specific IEC 559.

Remainder after floating point division and floor to integer cannot be exact. For a small negative nominator and a positive denominator, the resulting value looses much absolute accuracy in relation to the original value. Such an operation is therefore not included in ISO/IEC 10967-2.

See also the radian and the argument angular-unit normalisation operations (5.3.6.1, 5.3.7.1).

A.5.2.6 Square root and reciprocal square root operations

The inverses of squares are double valued, the two possible results having the same magnitude with opposite signs. For a non-zero result, ISO/IEC 10967-2 requires that each of the corresponding operations return a positive result.

There is no ambiguity in the result for $sqrt_F(x)$: the existence of an ambiguity would require that the corresponding mathematical function could yield a result exactly half-way between two successive floating point numbers. Such a number would require exactly $(p+1)$ digits for its exact representation. The square of such a number would require at least $(2p+1)$ digits, which could not equal the p-digit number x.

The extensions $sqrt_F(+\infty) = +\infty$ and $sqrt_F(\mathbf{-0}) = \mathbf{-0}$ are mandated by IEC 559. LIA-2 requires that these axioms hold for implementations which support infinities and signed zeros. However, it should be noted that while the second is harmless, the first may lead to erroneous results: a $+\infty$ generated by an addition or subtraction is just barely outside of the normalised range of numbers. Hence its square root would be well within the representable range. The possibility that LIA-2 should require that $sqrt_F(+\infty) =$ **undefined** was considered, but rejected because of the principle of regarding arguments as exact, even if they are not exact. In addition $sqrt_F(+\infty) = +\infty$ for is already required by IEC 559.

Note that the requirement that $sqrt_F(x) =$ **invalid**(**qNaN**) for x strictly less than zero is mandated by IEC 559. It follows that **NaN**s generated in this way represent imaginary values, which would become complex through addition and subtraction, and even imaginary infinities on multiplication by ordinary infinities.

The $rsqrt_F$ operation will increase performance for scaling a vector into a unit vector. Such an operation involves division of each component of the vector by the magnitude of the vector or, equivalently and with higher performance, multiplication by the reciprocal of the magnitude.

A.5.2.7 Support operations for extended floating point precision

These operations would typically be used to extend the precision of the highest level supported by the underlying hardware of an implementation.

The major motivation for including them in LIA-2 is to provide a capability for accurately evaluating residuals in an iterative procedure. The residuals give a measure of the error in the current solution. More important they can be used to estimate a correction to the current solution. The accuracy of the correction depends on the accuracy of the residuals. The residuals are calculated as a difference in which the number of leading digits cancelled increases as the accuracy of the solution increases. A doubled precision calculation of the residuals is usually adequate to produce a reasonably efficient iteration.

For the basic floating point arithmetic doubled precision operations, the high parts are calculated the corresponding floating point operations.

There is no intent to provide a set of operations suitable for the implementation of a complete package for the support of calculations at an arbitrarily high level of precision.

If $add_F(x, y)$ rounds to *nearest* then the high and low parts represent $x + y$ exactly.

The product of two numbers, each with p digits of precision, is always exactly representable in at most $2p$ digits. The high and low parts of the product will always represent the true product.

The remainder for division is more useful than a $2p$-digit approximation. The remainder will be exactly representable if the high part differs from the true quotient by less than one *ulp*. The true quotient can be constructed p digits at a time by division of the successive remainders by the divisor.

The remainder for square root is more useful than a low part for the same reason that the remainder is more useful for division. The remainder for the square root operation will be exactly representable only if the high part is correctly rounded to nearest, as is required by the specification for $sqrt_F$.

A.5.2.8 Extended precision multiply

This operation is intended for the case that there exist at least two floating point datatypes F and F', such that the product of two numbers of type F is always exactly representable in type F'.

To obtain higher precision for multiplication, in the absence of a suitable level of precision F', a programmer can exploit the paired mul_F and mul_lo_F operations.

A.5.2.9 Extended precision multiply and add

This operation should multiply using a $2p$-digit accumulator, add the third argument, with the result rounded by the rounding rule to the original p-digit level of precision.

A.5.2.10 Exact summation operation

This operation can be used in conjunction with doubled precision multiplication to generate an exact inner product. An important application is in the calculation of residuals for an iterative solution of a system of linear equations, $A * x = b$ where A is an n by n matrix and x and b are n-vectors. If x_0 is the current solution, then the correction u is given by $A * u = b - A * x_0$. The term $A * x_0$ is a vector of inner products.

A.5.3 Elementary transcendental floating point operations

A.5.3.1 Specification format

The terms "numerical function" and "mathematical function" are used to distinguish between a method for approximating a mathematical function and the approximated mathematical function itself.

The signature of an operation identifies the arithmetic datatypes for the input operands and the output produced by a operation. The datatypes in the signature of an operation also appear as subscripts to the name of the operation. For some operations the exceptional value **invalid** is produced only by input values of **−0**, $+\infty$, $-\infty$, or **sNaN**. For these operations the signature does not contain **invalid**. In general, LIA-2 does *not* specify operations in terms of identities like

$$power_F(x, y) = exp_F(mul_F(y, ln_F(x))$$

in order to avoid an implied requirement that a particular algorithm be used to implement the operation, an algorithm which in addition may result in less accuracy than may be otherwise attainable.

A.5.3.1.1 Maximum error requirements

$max\_error\_op_F$ measures the discrepancy between the computed value $op_F(x)$ and the true mathematical value $f(x)$ in ulps of the true value. The magnitude of the error bound is thus available to a program from the computed value $op_F(x)$. Note that for results at an exponent boundary for F, y, the error away from zero is in terms of $ulp_F(y)$, whereas the error toward zero is in terms of $ulp_F(y)/r_F$, which is the ulp of values slightly smaller in magnitude than y.

Within limits, accuracy and performance may be varied to best meet customer needs. Note also that LIA-2 does not prevent a vendor from offering two or more implementations of the various operations.

The operation specifications define the domain and range for the operations. The computational domain and range are more limited for the operations than for the corresponding mathematical functions because the arithmetic datatypes are subsets of $\mathcal{R}$ and $\mathcal{Z}$. Thus the actual domain of $exp_F(x)$ is approximately given by

$$\ln(fmin_F) \leq x \leq \ln(fmax_F)$$

The actual range extends over F, although there are values, $v \in F$, for which there is no $x \in F$ satisfying

$$exp_F(x) = v.$$

The numerical functions may produce any of the exceptional values **integer_overflow**, **floating_overflow**, **underflow**, **invalid**, **pole**, or **angle_too_big**.

The thresholds for the **integer_overflow**, **floating_overflow**, and **underflow** notifications are determined by the parameters defining the arithmetic datatypes.

The threshold for an **undefined** notification is determined by the domain of input arguments for which the mathematical function being approximated is defined.

The **pole** notification is the operation's counterpart of a mathematical pole of the mathematical function being approximated by the operation.

The threshold for **angle_too_big** is determined by the parameters $big\_angle\_r_F$ and $big\_angle\_u_F$ supplied by the implementation.

LIA-2 imposes a fairly tight bound on the maximum error allowed in the implementation of each operation. The tightest possible bound is given by requiring rounding to nearest, for which the accompanying performance penalty is often unacceptably high. LIA-2 requires rounding to nearest for only a few operations.

The parameters $max\_error\_op_F$ will be documented by the implementation for each such parameter required by LIA-2. A comparison of the values of these parameters with the values of the specified maximum value for each such parameter will give some indication of the "quality" of the routines provided. Further, a comparison of the values of this parameter for two versions of a frequently used operation will give some indication of the accuracy sacrifice made in order to gain performance.

Language bindings are free to modify the error limits provided in the specifications for the operations to meet the expected requirements of their users.

Material on the implementation of high accuracy operations is provided in for example [30, 32, 38].

A.5.3.1.2 The *trans_result* helper function

A.5.3.1.3 Sign requirements

A.5.3.1.4 Monotonicity requirements

A.5.3.1.5 IEC 559 special values

The signed zeros, infinities, and NaNs introduced in IEC 559, are implemented in many current implementations, and can be expected to become a standard part of floating point calculations. These special values can be generated as continuation values in such implementations, via literals for these values, and as the true result when appropriate.

It follows that they can occur as input to arithmetic operations on any implementation which supports them. Implementations which provide these special values may conform to IEC 559. Moreover, implementations which do not support these special values are required to document such alternative actions as they provide.

A report ([36]) issued by the ANSI X3J11 committee discusses possible ways of exploiting these features. The report identifies some of its suggestions as controversial and cites [32] as justification.

The next four clauses summarise the specifications of IEC 559 on the creation and propagation of signed zeros, infinities, and **NaN**s. They also include some discussion of material in [32, 33, 30].

IEC 559 regards 0 and $-\mathbf{0}$ as almost indistinguishable. The sign is supposed to indicate the direction of approach to zero. The sign is reliable for a zero generated by underflow in a multiplication or division operation. It is not reliable for a zero generated by an implied subtraction of two floating point numbers with the same value, for which case the zero is arbitrarily given a + sign. The phrase "implied subtraction" indicates either the addition of two oppositely signed numbers or the subtraction of two like signed numbers.

On occurrence of floating overflow or division of a non-zero number by zero, an implementation conforming to IEC 559 sets the appropriate status flag (if trapping is not enabled) and then continues execution with a result of $+\infty$ or $-\infty$.

IEC 559 states that the arithmetic of infinities is that associated with mathematical infinities. Thus, an infinity times, plus, minus, or divided by a non-zero floating point number yields an infinity for the result; no status flag is set and execution continues. These rules are not necessarily valid for infinities generated by overflow, thought they are valid if the infinitary arguments are exact.

NaNs are generated by invalid operations on infinities, 0/0, and the square root of a negative number (other than $-\mathbf{0}$). Thus **NaN**s can represent unknown real or complex values, as well as totally undefined values.

IEC 559 requires that the result of any of its basic operations with one or more **NaN** inputs shall be a **NaN**. This principle is not extended to the numerical functions by [32, 36].

The controversial specifications in [36] are based on an assumption that all of these special operands represent finite non-zero real-valued numbers; see [32, 33].

The LIA-2 policy for dealing with signed zeros, infinities, and **NaN**s is as follows:

a) The output is a **NaN** for any operation for which one (or more) inputs is a **NaN**. There is no notification.

b) If a mathematical function $h(x)$ is such that $h(0) = 0$, the corresponding operation $op_F(x)$ returns x if $x \in \{0, -\mathbf{0}\}$ and h has a positive derivative at 0, and $op_F(x)$ returns $neg_F(x)$ if $x \in \{0, -\mathbf{0}\}$ and h has a negative derivative at 0.

c) For an input value, x, of 0, **−0**, $+\infty$, or $-\infty$, the output value of the operation $op(x)$ is

$$\lim_{z \to x} h(z)$$

where the an approach to zero if from the positive side if $x = 0$, and the approach is from the negative side if $x =$ **−0**.

There is no notification if the limit exists, is finite, and is path independent. The returned value is $+\infty$ or $-\infty$ if the limiting value is unbounded, and the approach is towards an infinity. The returned value is **pole**$(+\infty)$ or **pole**$(-\infty)$ if the limiting value is unbounded, and the approach is towards zero.

If the limit does not exist the value returned is **invalid**, and a notification occurs, with a continuation value of **qNaN** if appropriate.

A.5.3.2 Hypotenuse operation

The $hypot_F$ operation can produce an overflow only if both arguments have magnitudes very close to the overflow threshold. Care must be taken in its implementation to either avoid or properly handle overflows and underflows which might occur in squaring the arguments. The function approximated by this operation is mathematically equivalent to complex absolute value, which is needed in the calculation of the modulus and argument of a complex number. It is important for this application that an implementation satisfy the constraint on the magnitude of the result returned.

LIA-2 does not follow the recommendations in [32] and in [33] that

$hypot_F(+\infty, \mathbf{qNaN}) = +\infty$
$hypot_F(-\infty, \mathbf{qNaN}) = +\infty$
$hypot_F(\mathbf{qNaN}, +\infty) = +\infty$
$hypot_F(\mathbf{qNaN}, -\infty) = +\infty$

which are based on the claim that a **qNaN** represents an (unknown) real valued number. This claim is not always valid, though it may sometimes be.

A.5.3.3 Operations for exponentiations and logarithms

For all of the exponentiation operations, overflow occurs for sufficiently large values of the argument(s).

There is a problem for $power_F(x, y)$ if both x and y are zero:

- Ada raises an exception for the operation that is close in semantics to $power_F$ when both arguments are zero, in accordance with the fact that 0^0 is mathematically undefined.
- The X/OPEN Portability Guide specifies for `pow(0,0)` a return value of 1, and no notification. This specification agrees with the recommendations in [30, 32, 33, 36].

The specification in LIA-2 follows Ada, and returns **invalid** for $power_F(0, 0)$ (with the continuation value 1), because of the risks inherent in returning a result which might be inappropriate for the application at hand.

The specifications for input of $+\infty$ or $-\infty$ are non-controversial, and are consistent with the behaviour of the mathematical function x^y.

The arguments of $power_F$ are floating point numbers. No special treatment is provided for integer floating point values, which may be approximate. The cases for integer values of the arguments are covered by the operations $power_{FI}$ and $power_I$.

The result of the $power_F$ operation is **invalid** for negative values of the base x. The reason is that the floating point exponent y might imply an implicit extraction of an even root of x, which would have a complex value for negative x. This constraint is explicit in Ada, and is widely imposed in existing numerical packages provided by vendors.

Along any curve defined by $y = k/ln(x)$ the mathematical function x^y has the value e^k. It follows that some of the limiting values for x^y depend on the choice of k, and hence are undefined, as indicated in the specification.

There is an accuracy problem with an algorithm based on the following identity:

$$x^y = r_F^{y*\log_{r_F}(x)}$$

The integer part of the product $y * \log_{r_F}(x)$ defines the exponent of the result and the fractional part defines the reduced argument. If the exponent is large, and one calculates p_F digits of this intermediate result, there will be fewer than p_F digits for the fraction. Thus, in order to obtain a reduced argument accurately rounded to p digits, it may be necessary to calculate an approximation to $y * \log_{r_F}(x)$ to a few more than $\log_{r_F}(emax_F) + p_F$ base r_F digits.

The special exponential operations, corresponding to 2^x and 10^x, have specifications which are minor variations on those for $exp_F(x)$. Accuracy and performance can be increased if they are specially coded, rather than evaluated as $exp_F(mul_F(x, ln_F(2)))$ or $power_F(2, x)$.

Similar comments hold for the base 2 and base 10 logarithmic operations.

A.5.3.4 Operations for hyperbolics and inverse hyperbolics

The hyperbolic sine operation, $sinh_F(x)$, will overflow if $|x|$ is in the immediate neighbourhood of $\ln(2 * fmax)$, or greater.

The hyperbolic cosine operation, $cosh_F(x)$, will overflow if $|x|$ is in the immediate neighbourhood of $\ln(2 * fmax)$, or greater.

The hyperbolic cotangent operation, $coth_F(x)$, has a pole at $x = 0$.

The inverse of cosh is double valued, the two possible results having the same magnitude with opposite signs. The value returned by $arccosh_F$ is always greater than or equal to 1.

The inverse hyperbolic tangent operation $arctanh_F(x)$ has poles at $x = +1$ and at $x = -1$.

The inverse hyperbolic cotangent operation $arccoth_F(x)$ has poles at $x = +1$ and at $x = -1$.

A.5.3.5 Introduction to operations for trigonometrics

A.5.3.6 Operations for radian trigonometrics and inverse radian trigonometrics

The real trigonometric functions $\sin(x)$, $\cos(x)$, $\tan(x)$, $\cot(x)$, $\sec(x)$, and $\csc(x)$ are all periodic in the (real) argument x. The period for sin, cos, sec, and csc is $2 * \pi$ radians (360 degrees). The period for tan and cot is π radians (180 degrees).

There are three ways in which a limitation on the accuracy of a trigonometric operation can be identified:

- The first is related to the fact that the density of floating point values gets sparser as the magnitude of the values increases. For arguments known to be exact, sparsity implies no accuracy problems.

  For a trigonometric operation, the number of floating point values per period gets sparser as the magnitude of the argument increases. Hence, for approximately computed arguments,

there is a maximum argument for which the sparsity will pose no problem. The Ada standard suggests the value $r_F^{\lfloor p_F/2 \rfloor}$ for radian reduction.

– For reduction of an argument given in radians, implementations use one or several approximate value(s) of π (or of a multiple of π), valid to, say, n digits. It follows that the division implied in the argument reduction cannot be valid to more than n digits, which implies a maximum absolute angle value for which the reduction yields an accurate reduced angle value.

ISO/IEC 10967-2 defines two parameters for the identification of the maximum argument for which the trigonometric operations are guaranteed to satisfy the accuracy requirements: one, $big\_arg\_r_F$, refers to operations with radian arguments. The other, $big\_arg\_u_F$, refers to operations with angular unit (including degree) arguments.

An implementation must support a maximum argument parameter for which a set of trigonometric operations is implemented. The value of each of the parameters is determined by the implementation.

The argument reduction techniques for radians described in [38] avoid the inaccuracies mentioned above. Moreover, at least for currently available floating point implementations, this techniques can produce p digit reduced arguments with an error bound of $ulp/2$.

All six functions have an essential singularity at infinity. In addition

– tan and sec have *poles* at odd multiples of $\pi/2$ radians (90 degrees).

– cot and csc have *poles* at multiples of π radians (180 degrees).

All four of the corresponding operations with poles may produce **floating_overflow** for arguments sufficiently close to the poles of the functions.

The **pole** notifications cannot occur if a non-zero argument is in radians because π is not representable in F, except when the pole occurs at 0. For the angular unit argument trigonometric operations a continuation value of $+\infty$ has been chosen arbitrarily for a **pole** which occurs for a positive argument.

The operations may produce **underflow** for arguments sufficiently close to their zeros.

For a denormalised argument x, the sin_F and tan_F operations can return x for the result, with very high accuracy. Similarly, for a denormalised argument, cos_F and sec_F can return a result of 1.0 with very high accuracy.

At present only Ada specifies trigonometric operations with angular unit argument. ISO/IEC 10967-2 has adopted angular unit argument operations in order to encourage uniformity among languages which might include such operations in the future. The angular units in T appear to be particularly important and have therefore been given a tighter error bound requirement. An implementation can of course have the same (tighter) error bound for all angular units.

Few languages require the functions with the argument in degrees. However, they are almost universally provided for Fortran.

The tan_F operation produces no **pole** notifications if its argument is an element of F. The reason is that the poles of $\tan(x)$ are at odd multiples of $\pi/2$, which are not representable in F.

The mathematical cotangent function has a pole at the origin. For a system which supports signed zeros and infinities, the continuation values are $+\infty$ and $-\infty$ for arguments of 0 and **−0** respectively.

Although the mathematical function sec has poles at odd multiples of $\pi/2$, the sec_F operation will not generate them because such arguments are not representable in F. The situation is the same as for the tangent function in radians.

The corresponding mathematical functions are multiple valued. They are rendered single valued by defining a principal value range. This range is closely related to a branch cut in the complex plane for the corresponding complex function. Among the numerical functions this branch cut is "visible" only for the arctan2 operation.

The principal value ranges are not uniquely determined.

The **invalid** and **underflow** notifications are the only notifications produced by the inverse trigonometric functions.

The arc function has a branch cut along the negative real axis. For $x < 0$ the function has a discontinuity from $-\pi$ to $+\pi$ as y passes through zero from negative to positive values. Thus for $x < 0$, systems supporting signed zeros can handle the discontinuity as follows:

$arc_F(x, \mathbf{-0}) = -nearest_F(\pi)$
$arc_F(x, 0) = +nearest_F(\pi)$

The values given for the operation $arc_F(x, y)$ for the four combinations of signed zeros for x and y are those given in [32]. There is a problem for input values of $+\infty$ or $-\infty$ for this operation. The following table of values is given in [32] for the value of $arc_F(x, y)$ with at least one of the arguments infinite:

Infinite arguments		
x	y	$arc_F(x, y)$
$+\infty$	$b \geq 0$	0
$+\infty$	$+\infty$	$\pi/4$
$b \leq 0$	$+\infty$	$\pi/2$
$-\infty$	$+\infty$	$3 * \pi/4$
$-\infty$	$b \geq 0$	π
$-\infty$	$b \leq \mathbf{-0}$	$-\pi$
$-\infty$	$-\infty$	$-3 * \pi/4$
$b \leq \mathbf{-0}$	$-\infty$	$-\pi/2$
$+\infty$	$-\infty$	$-\pi/4$
$+\infty$	$b \leq \mathbf{-0}$	$\mathbf{-0}$

where b represents a finite number.

If one of x and y is infinite and the other is finite, the result tabulated is consistent with that obtained by a conventional limiting process. ISO/IEC 10967-2 provides these results.

However, the results of $\pi/4$, $-\pi/4$, $3 * \pi/4$, and $-3 * \pi/4$ corresponding to infinite values for both x and y, are of questionable validity.

A.5.3.7 Operations for argument angular unit trigonometrics and inverse argument angular unit trigonometrics

If the angular unit argument, u, is such that $u/4 \in F$, the $tanu_F$ operation has poles at odd multiples of $u/4$. This is the case for degrees ($u = 360$).

As for $tanu_F$, if the angular unit argument, u, is such that $u/4 \in F$ the $secu_F$ operation has poles at odd multiples of $u/4$.

The same comments hold for the $arcu_F$ operation as for arc_F operation, except that the discontinuity in the mathematical function is from $-u/2$ to $+u/2$.

A.5.3.8 Operations for degree trigonometrics and inverse degree trigonometrics

Few languages require the trigonometric operations with angles in degrees. However, they are almost universally provided for Fortran. Performance can probably be gained by implementing them as functions of the single argument x measured in degrees, rather than using the two argument forms.

A.5.3.9 Operations for angular-unit conversions

A.5.4 Conversion operations

Clause 5.2 of ISO/IEC 10967-1 covers conversions from an integer type to another integer type and to a floating point type.

A.6 Notification

The reason for omitting notification for underflow for an operation for which the corresponding mathematical function satisfies $f(x) \approx x$ if $|x| \leq fminN$ is as follows: such an underflow can happen only if the input operand x is a denormalised number. Hence a notification must have already been returned when its denormalisation was created. Nothing is gained by "repeating" the notification, particularly since the calculation of $f(x)$ is very accurate (relative error much less than $epsilon_F/2$).

A.6.1 Continuation values

An implementation which supports notification by a recording of indicators (ISO/IEC 10967-1, clause 6.1.2) must supply values to be used if execution is to be continued following occurrence of a **floating_overflow**, **underflow**, or **undefined**. For systems which also support signed zeros, infinities and **NaN**s, LIA-2 specifies how these entities are used for continuation values. Other implementations must supply continuation values and document the values selected.

A.7 Relationship with language standards

A.8 Documentation requirements

Annex B
(informative)

Partial conformity

If an implementation of an operation fulfills all relevant requirements according to the normative text in LIA-2, except the ones relaxed in this annex, the implementation of that operation is said to *partially conform* to LIA-2.

LIA-2 has the following max error requirements for full conformity.

$max\_error\_hypot_F \in [0.5, 1]$

$max\_error\_exp_F \in [0.5, 1.5 * rnd\_error_F]$
$max\_error\_power_F \in [max\_error\_exp_F, 2 * rnd\_error_F]$

$max\_error\_sinh_F \in [0.5, 2 * rnd\_error_F]$
$max\_error\_tanh_F \in [max\_error\_sinh_F, 2 * rnd\_error_F]$

$max\_error\_sin_F \in [0.5, 1.5 * rnd\_error_F]$
$max\_error\_tan_F \in [max\_error\_sin_F, 2 * rnd\_error_F]$

$max\_error\_sinu_F : F \to F \cup \{\textbf{invalid}\}$
$max\_error\_tanu_F : F \to F \cup \{\textbf{invalid}\}$

For $u \in G_F$, the $max\_error\_sinu_F(u)$ parameter shall be in the interval $[max\_error\_sin_F, 2]$. The $max\_error\_sinu_F(u)$ parameter shall be equal to $max\_error\_sin_F$ if $u \in T$.

For $u \in G_F$, the $max\_error\_tanu_F(u)$ parameter shall be in the interval $[max\_error\_tan_F, 4]$. The $max\_error\_tanu_F(u)$ parameter shall be equal to $max\_error\_tan_F$ if $u \in T$.

In a partially conforming implementation the max error parameters may be greater than what is specified by LIA-2. The max error parameter values given in an implementation must still adequately reflect the accuracy of the relevant operations, if a claim of partial conformity is made. A partially conforming implementation must document which max error parameters have greater values than specified by LIA-2.

LIA-2 has a number of extra accuracy requirements in section 5.3. These are detailed in the paragraphs beginning "Further requirements on the op^*_F approximation helper function". In a partially conforming implementation these further requirements need not be fulfilled. The values returned must still be within the max error bounds that are given by the max error parameters, if a claim of partial conformity is made. A partially conforming implementation must document which 'further requirements' that are not fulfilled by the implementation.

Annex C
(informative)

Example bindings for specific languages

This annex describes how a computing system can simultaneously conform to a language standard (or publicly available specification) and to ISO/IEC 10967-2. It contains suggestions for binding the "abstract" operations specified in ISO/IEC 10967-2 to concrete language syntax.

Portability of programs can be improved if two conforming LIA-2 systems using the same language agree in the manner with which they adhere to LIA-2. For instance, LIA-2 requires that the parameter $big\_angle\_r_F$ be provided (if any conforming , but if one system provides it by means of the identifier `BigAngle` and another by the identifier `MaxAngle`, portability is impaired. Clearly, it would be best if such names were defined in the relevant language standards or binding standards, but in the meantime, suggestions are given here to aid portability.

The following clauses are suggestions rather than requirements because the areas covered are the responsibility of the various language standards committees. Until binding standards are in place, implementors can promote "de facto" portability by following these suggestions on their own.

The languages covered in this annex are

Ada
Basic
C and C++
Fortran
Java
ISLisp and Common Lisp
Modula-2
Pascal and Extended Pascal

This list is not exhaustive. Other languages and other computing devices (like 'scientific' calculators, and database 'query languages') are suitable for conformity to ISO/IEC 10967-2.

In this annex, the parameters, operations, and exception behaviour of each language are examined to see how closely they fit the requirements of ISO/IEC 10967-2. Where parameters, constants, or operations are not provided by the language, names and syntax are suggested.

This annex describes only the language-level support for ISO/IEC 10967-2. An implementation that wishes to conform must ensure that the underlying hardware and software is also configured to conform to ISO/IEC 10967-2 requirements.

A complete binding for ISO/IEC 10967-2 will include, or refer to, a binding for ISO/IEC 10967-1. In turn, a complete binding for the ISO/IEC 10967-1 will include a binding for IEC 559. Such a joint LIA-2/LIA-1/IEC 559 binding should be developed as a single binding standard. To avoid conflict with ongoing development, only the ISO/IEC 10967-2 specific portions of such a binding are presented in this annex.

C.1 General comments

Most language standards permit an implementation to provide, by some means, the parameters and operations required by ISO/IEC 10967-2 that are not already part of the language. The method for accessing these additional parameters and operations depends on the implementation and language, and is not specified in ISO/IEC 10967-2 nor exemplified in this annex. It could

include external subroutine libraries; new intrinsic functions supported by the compiler; constants and functions provided as global "macros"; and so on.

Most language standards do not constrain the accuracy of elementary numerical functions, or specify the subsequent behaviour after a serious arithmetic violation occurs.

In the event that there is a conflict between the requirements of the language standard and the requirements of ISO/IEC 10967-2, the language binding standard should clearly identify the conflict and state its resolution of the conflict.

C.2 Ada

The programming language Ada is defined by ISO/IEC 8652:1995, *Information Technology – Programming Languages – Ada* [6].

An implementation should follow all the requirements of LIA-2 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the LIA-2 for that operation. For each of the marked items a suggested identifier is provided.

The Ada datatype `Boolean` corresponds to the ISO/IEC 10967-1 datatype *Boolean*.

Every implementation of Ada has at least one integer datatype, and at least one floating point datatype. The notations *INT* and *FLT* are used to stand for the names of one of these datatypes in what follows.

The additional integer operations are listed below, along with the syntax used to invoke them:

$max_I(x,y)$	*INT*'`Max(`x`, `y`)`	
$min_I(x,y)$	*INT*'`Min(`x`, `y`)`	
$max\_seq_I(xs)$	`Max(`xs`)`	†
$min\_seq_I(xs)$	`Min(`xs`)`	†
$dim_I(x,y)$	`Dim(`x`, `y`)`	†
$power_I(x,y)$	x `**` y	
$sqrt_I(x)$	`Sqrt(`x`)`	†
$divides_I(x,y)$	`Divides(`x`, `y`)`	†
$even_I(x)$	`Even(`x`)`	†
$odd_I(x)$	`Odd(`x`)`	†
$gcd_I(x,y)$	`Gcd(`x`, `y`)`	†
$lcm_I(x,y)$	`Lcm(`x`, `y`)`	†
$gcd\_seq_I(xs)$	`Gcd(`xs`)`	†
$lcm\_seq_I(xs)$	`Lcm(`xs`)`	†
$add\_wrap_I(x,y)$	`Add_wrap(`x`, `y`)`	†
$add\_ov_I(x,y)$	`Add_over(`x`, `y`)`	†
$sub\_wrap_I(x,y)$	`Sub_wrap(`x`, `y`)`	†
$sub\_ov_I(x,y)$	`Sub_over(`x`, `y`)`	†
$mul\_wrap_I(x,y)$	`Mul_wrap(`x`, `y`)`	†
$mul\_ov_I(x,y)$	`Mul_over(`x`, `y`)`	†

where x and y are expressions of type *INT* and where xs is an expression of type `array of` *INT*.

The additional basic floating point operations are listed below, along with the syntax used to invoke them:

$max_F(x,y)$	FLT'`Max(`x`, `y`)`	
$min_F(x,y)$	FLT'`Min(`x`, `y`)`	
$mmax_F(x,y)$	`Mmax(`x`, `y`)`	†
$mmin_F(x,y)$	`Mmin(`x`, `y`)`	†
$max\_seq_F(xs)$	`Max(`xs`)`	†
$min\_seq_F(xs)$	`Min(`xs`)`	†
$mmax\_seq_F(xs)$	`Mmax(`xs`)`	†
$mmin\_seq_F(xs)$	`Mmin(`xs`)`	†
$dim_F(x,y)$	`Dim(`x`, `y`)`	†
$rounding_F(x)$	FLT'`Unbiased_Rounding(`x`)`	
$floor_F(x)$	FLT'`Floor(`x`)`	
$ceiling_F(x)$	FLT'`Ceiling(`x`)`	
$rounding\_rest_F(x)$	(x - FLT'`Unbiased_Rounding(`x`)`)	
$floor\_rest_F(x)$	(x - FLT'`Floor(`x`)`)	
$ceiling\_rest_F(x)$	(x - FLT'`Ceiling(`x`)`)	
$sqrt_F(x)$	`Sqrt(`x`)`	
$rsqrt_F(x)$	`Rsqrt(`x`)`	†
$irem_F(x,y)$	FLT'`Remainder(`x`, `y`)`	
$add\_lo_F(x,y)$	`Add_low(`x`, `y`)`	†
$sub\_lo_F(x,y)$	`Sub_low(`x`, `y`)`	†
$mul\_lo_F(x,y)$	`Mul_low(`x`, `y`)`	†
$div\_rest_F(x,y)$	`Div_rest(`x`, `y`)`	†
$sqrt\_rest_F(x)$	`Sqrt_rest(`x`)`	†
$add3_F(x,y,z)$	`Add(`x`, `y`, `z`)`	†
$add3\_mid_F(x,y,z)$	`AddMid(`x`, `y`, `z`)`	†
$mul\_add_F(x,y,z)$	`Mul_add(`x`, `y`, `z`)`	†
$dprod_{F\to F'}(x,y)$	`Prod(`x`, `y`)`	†
$sum_F(xs)$	`Sum(`xs`)`	†

where x, y and z are expressions of type FLT, and where xs is an expression of type **`array of`** FLT.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$max\_err\_hypot_F$	`Err_hypotenuse(`x`)`	†
$max\_err\_exp_F$	`Err_exp(`x`)`	†
$max\_err\_power_F$	`Err_power(`x`)`	†
$max\_err\_sinh_F$	`Err_sinh(`x`)`	†
$max\_err\_tanh_F$	`Err_tanh(`x`)`	†
$big\_angle\_r_F$	`Big_radian_angle(`x`)`	†
$max\_err\_sin_F$	`Err_sin(`x`)`	†
$max\_err\_tan_F$	`Err_tan(`x`)`	†
$big\_angle\_u_F$	`Big_angle(`x`)`	†
$max\_err\_sinu_F(u)$	`Err_sin_cycle(`u`)`	†
$max\_err\_tanu_F(u)$	`Err_tan_cycle(`u`)`	†

where x and u are expressions of type *FLT*. Several of the parameter functions are constant for each type (and library), the argument is then used only to differentiate among the floating point types.

$hypot_F(x,y)$	`Hypotenuse(`x`, `y`)`	†
$exp_F(x)$	`Exp(`x`)`	
$expm1_F(x)$	`ExpM1(`x`)`	†
$power_F(b,y)$	b `**` y	
$power_{FI}(b,z)$	b `**` z	
$powerm1_F(b,y)$	`PowerM1(`b`, `y`)`	†
$exp2_F(x)$	`Exp2(`x`)`	†
$exp10_F(x)$	`Exp10(`x`)`	†
$ln_F(x)$	`Log(`x`)`	
$ln1p_F(x)$	`Log1P(`x`)`	†
$log_F(b,x)$	`Log(`x`, `b`)`	
$log1p_F(b,x)$	`Log1P(`x`, `b`)`	†
$log2_F(x)$	`Log2(`x`)`	†
$log10_F(x)$	`Log10(`x`)`	†
$sinh_F(x)$	`Sinh(`x`)`	
$cosh_F(x)$	`Cosh(`x`)`	
$tanh_F(x)$	`Tanh(`x`)`	
$coth_F(x)$	`Coth(`x`)`	
$sech_F(x)$	`Sech(`x`)`	†
$csch_F(x)$	`Csch(`x`)`	†
$arcsinh_F(x)$	`Arcsinh(`x`)`	
$arccosh_F(x)$	`Arccosh(`x`)`	
$arctanh_F(x)$	`Arctanh(`x`)`	
$arccoth_F(x)$	`Arccoth(`x`)`	
$arcsech_F(x)$	`Arcsech(`x`)`	†
$arccsch_F(x)$	`Arccsch(`x`)`	†
$rad\_nearest\_axis_F(x)$	`Rad_nearest_axis(`x`)`	†
$rad\_offset\_axis_F(x)$	`Rad_offset_axis(`x`)`	†
$rad_F(x)$	`Rad(`x`)`	†
$sin_F(x)$	`Sin(`x`)`	
$cos_F(x)$	`Cos(`x`)`	
$tan_F(x)$	`Tan(`x`)`	
$cot_F(x)$	`Cot(`x`)`	
$sec_F(x)$	`Sec(`x`)`	†
$csc_F(x)$	`Csc(`x`)`	†
$arcsin_F(x)$	`Arcsin(`x`)`	
$arccos_F(x)$	`Arccos(`x`)`	
$arc_F(x,y)$	`Arctan(`y`, `x`)`, or `Arccot(`x`, `y`)`	
$arctan_F(x)$	`Arctan(`x`)`	
$arccot_F(x)$	`Arccot(`x`)`	

$arcctg_F(x)$	(Sign(x)*Arccot(Abs(x)))	
$arcsec_F(x)$	Arcsec(x)	†
$arccsc_F(x)$	Arccsc(x)	†
$cycle\_nearest\_axis_F(u,x)$	Cycle_nearest_axis(x,u)	†
$cycle\_offset\_axis_F(u,x)$	Cycle_offset_axis(x,u)	†
$cycle_F(u,x)$	Cycle(x,u)	†
$sinu_F(u,x)$	Sin(x,u)	
$cosu_F(u,x)$	Cos(x,u)	
$tanu_F(u,x)$	Tan(x,u)	
$cotu_F(u,x)$	Cot(x,u)	
$secu_F(u,x)$	Sec(x,u)	†
$cscu_F(u,x)$	Csc(x,u)	†
$arcsinu_F(u,x)$	Arcsin(x,u)	
$arccosu_F(u,x)$	Arccos(x,u)	
$arcu_F(u,x,y)$	Arctan(y,x,u) or Arccot(x,y,u)	
$arctanu_F(u,x)$	Arctan(x, Cycle=>u)	
$arccotu_F(u,x)$	Arccot(x, Cycle=>u)	
$arcctgu_F(u,x)$	(Sign(x)*Arccot(Abs(x), Cycle=>u))	
$arcsecu_F(u,x)$	Arcsec(x,u)	†
$arccscu_F(u,x)$	Arccsc(x,u)	†
$deg\_nearest\_axis_F(u,x)$	Cycle_nearest_axis(x, 360.0)	†
$deg\_offset\_axis_F(u,x)$	Cycle_offset_axis(x, 360.0)	†
$deg_F(x)$	Cycle(x, 360.0)	†
$sind_F(x)$	Sin(x, 360.0)	
$cosd_F(x)$	Cos(x, 360.0)	
$tand_F(x)$	Tan(x, 360.0)	
$cotd_F(x)$	Cot(x, 360.0)	
$secd_F(x)$	Sec(x, 360.0)	†
$cscd_F(x)$	Csc(x, 360.0)	†
$arcsind_F(x)$	Arcsin(x, 360.0)	
$arccosd_F(x)$	Arccos(x, 360.0)	
$arcd_F(x,y)$	Arctan(y,x, 360.0), or Arccot(x,y, 360.0)	
$arctand_F(x)$	Arctan(x, Cycle=>360.0)	
$arccotd_F(x)$	Arccot(x, Cycle=>360.0)	
$arcctgd_F(x)$	(Sign(x)*Arccot(Abs(x), Cycle=>360.0))	
$arcsecd_F(x)$	Arcsec(x, 360.0)	†
$arccscd_F(x)$	Arccsc(x, 360.0)	†
$rad\_to\_cycle_F(x,u)$	Rad_to_cycle(x, u)	†
$cycle\_to\_rad_F(u,x)$	Cycle_to_rad(u, x)	†
$cycle\_to\_cycle_F(u,x,v)$	Cycle_to_cycle(u, x, v)	†
$rad\_to\_deg_F(x)$	Rad_to_deg(x)	†
$deg\_to\_rad_F(x)$	Deg_to_rad(x)	†
$deg\_to\_cycle_F(x,v)$	Deg_to_cycle(x, v)	†

$cycle\_to\_deg_F(u,x)$	`Cycle_to_deg(`u`, `x`)`	†

where b, x, y, u, and v are expressions of type FLT, and z is an expressions of type INT.

Type conversions in Ada are always explicit and usually use the destination datatype name as the name of the conversion function. Few of them fullfill the requirements in this standard, however.

$cvt_{I\to I'}(x)$	$INT2$`(`x`)`	
$cvt_{I\to I'}(x)$	`Get(`s`,`n`,`w`)`	convert from string s
$cvt_{I\to I'}(x)$	`Put(`s`,`x`,`$base$`?)`	convert to string s
$cvt_{I\to I'}(x)$	`Get(`f`?,`n`,`w`?)`	input from text file f
$cvt_{I\to I'}(x)$	`Put(`f`?,`x`,`w`?,`$base$`?)`	output to textfile f
$rounding_{F\to I}(y)$	INT`(`FLT`'Unbiased_Rounding(`y`))`	
$floor_{F\to I}(y)$	INT`(`FLT`'Floor(`y`))`	
$ceiling_{F\to I}(y)$	INT`(`FLT`'Ceiling(`y`))`	
$cvt^n_{I\to F}earest(x)$	FLT`_Nearest(`x`)`	†
$cvt^d_{I\to F}own(x)$	FLT`_Down(`x`)`	†
$cvt^u_{I\to F}p(x)$	FLT`_Up(`x`)`	†
$cvt^n_{F\to F'}earest(y)$	$FLT2$`(`y`)`	
$cvt^n_{F\to F'}earest(y)$	`Get(`s`,`n`,`w`)`	convert from string s
$cvt^n_{F\to F'}earest(y)$	`Put(`s`,`x`,`a`?,`e`?)`	convert to string s
$cvt^n_{F\to F'}earest(y)$	`Get(`f`?,`m`,`w`?)`	input from text file f
$cvt^n_{F\to F'}earest(y)$	`Put(`f`?,`x`,`i`?,`a`?,`e`?)`	output to text file f
$cvt^d_{F\to F'}own(y)$	$FLT2$`_Down(`y`)`	†
$cvt^d_{F''\to F}own(y)$	`Get_Down(`s`,`m`,`w`)`	convert from string s
$cvt^d_{F\to F''}own(y)$	`Put_Down(`s`,`x`,`a`?,`e`?)`	convert to string s
$cvt^u_{F\to F'}p(y)$	$FLT2$`_Up(`y`)`	†
$cvt^u_{F''\to F}p(y)$	`Get_Up(`s`,`m`,`w`)`	†convert from string s
$cvt^u_{F\to F''}p(y)$	`Put_Up(`s`,`x`,`a`?,`e`?)`	†convert to string s
$cvt^n_{F\to D}earest(y)$	FXD`(`y`)`	
$cvt^n_{F\to D'}earest(y)$	`Put(`s`,`y`,`a`?,`e`?)`	
$cvt^n_{F\to D'}earest(y)$	`Put(`f`?,`y`,`i`?,`a`?,`e`?)`	
$cvt^d_{F\to D}own(y)$	FXD`_Down(`y`)`	†
$cvt^d_{F\to D'}own(y)$	`Put_Down(`s`,`x`,`a`?,`e`?)`	†
$cvt^u_{F\to D}p(y)$	FXD`_Up(`y`)`	†
$cvt^u_{F\to D'}p(y)$	`Put_Up(`s`,`x`,`a`?,`e`?)`	†
$cvt^n_{D\to F}earest(z)$	FLT`(`z`)`	
$cvt^n_{D'\to F}earest(z)$	`Get(`s`,`n`,`w`?)`	
$cvt^n_{D'\to F}earest(z)$	`Get(`f`?,`n`,`w`?)`	
$cvt^d_{D\to F}own(z)$	FLT`_Down(`z`)`	†
$cvt^d_{D'\to F}own(z)$	`Get_Down(`s`,`m`,`w`)`	†
$cvt^u_{D\to F}p(z)$	FLT`_Up(`z`)`	†
$cvt^u_{D'\to F}p(z)$	`Put_Up(`s`,`x`,`a`?,`e`?)`	†

where x is an expressions of type *INT*, y is an expressions of type *FLT*, and z is an expressions of type *FXD*, where *FXD* is a fixed point type. *INT2* is the integer datatype that corresponds to I'. A ? above indicates that the parameter is optional.

Ada provides non-negative numerals for all its integer and floating point types. The default base is 10, but all bases from 2 to 16 can be used. There is no differentiation between the numerals for different floating point types, nor between numerals for different integer types, but integer numerals (without a point) cannot be used for floating point types, and 'real' numerals (with a point) cannot be used for integer types. Integer numerals can have an exponent part though. The details are not repeated in this example binding, see ISO/IEC 8652:1995, clause 2.4 Numeric Literals, clause 3.5.4 Integer Types, and clause 3.5.6 Real Types.

Numerals for infinity...

String formats for numerals (same as numerals in Ada programs?).

C.3 BASIC

The programming language BASIC is defined by ISO/IEC 10279:1991, *Information Technology – Programming Languages – Full BASIC* [12].

An implementation should follow all the requirements of LIA-2 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the ISO/IEC 109672 for that operation. For each of the marked items a suggested identifier is provided.

The BASIC datatype `????` corresponds to the LIA-1 datatype *Boolean*.

The additional integer operations are listed below, along with the syntax used to invoke them:

$min_I(x,y)$	`Min(`x`, `y`)`	†
$max_I(x,y)$	`Max(`x`, `y`)`	†
$min\_seq_I(xs)$	`Min(`xs`)`	†
$max\_seq_I(xs)$	`Max(`xs`)`	†
$dim_I(x,y)$	`Dim(`x`, `y`)`	†
$sqrt_I(x)$	`Sqrt(`x`)`	†
$power_I(x,y)$	x `**` y	†
$divides_I(x,y)$	`Divides(`x`, `y`)`	†
$even_I(x)$	`Even(`x`)`	†
$odd_I(x)$	`Odd(`x`)`	†
$gcd_I(x,y)$	`Gcd(`x`, `y`)`	†
$lcm_I(x,y)$	`Lcm(`x`, `y`)`	†
$gcd\_seq_I(xs)$	`Gcd(`xs`)`	†
$lcm\_seq_I(xs)$	`Lcm(`xs`)`	†
$add\_wrap_I(x,y)$	`Add_wrap(`x`, `y`)`	†
$add\_ov_I(x,y)$	`Add_over(`x`, `y`)`	†
$sub\_wrap_I(x,y)$	`Sub_wrap(`x`, `y`)`	†
$sub\_ov_I(x,y)$	`Sub_over(`x`, `y`)`	†
$mul\_wrap_I(x,y)$	`Mul_wrap(`x`, `y`)`	†
$mul\_ov_I(x,y)$	`Mul_over(`x`, `y`)`	†

where x and y are expressions of type *INT* and where xs is an expression of type `array of` *INT*.

The additional non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$min_F(x,y)$	`Min(`x`, `y`)`	†
$max_F(x,y)$	`Max(`x`, `y`)`	†
$min\_seq_F(xs)$	`Min(`xs`)`	†
$max\_seq_F(xs)$	`Max(`xs`)`	†
$rounding_F(x)$	`Rounding(`x`)`	†
$floor_F(x)$	`Floor(`x`)`	
$ceiling_F(x)$	`Ceiling(`x`)`	†
$dim_F(x,y)$	`Dim(`x`, `y`)`	†
$add3_F(x,y,z)$	`Add(`x`, `y`, `z`)`	†
$sum_F(xs)$	`Sum(`xs`)`	†
$dprod_{F\to F'}(x,y)$	`Prod(`x`, `y`)`	†
$mul\_add_F(x,y,z)$	`Mul_add(`x`, `y`, `z`)`	†
$irem_F(x,y)$	`Remainder(`x`, `y`)`	†
$sqrt_F(x)$	`Sqrt(`x`)`	
$rsqrt_F(x)$	`Rsqrt(`x`)`	†
$add\_lo_F(x,y)$	`Add_low(`x`, `y`)`	†
$sub\_lo_F(x,y)$	`Sub_low(`x`, `y`)`	†
$mul\_lo_F(x,y)$	`Mul_low(`x`, `y`)`	†
$div\_rest_F(x,y)$	`Div_rest(`x`, `y`)`	†
$sqrt\_rest_F(x)$	`Sqrt_rest(`x`)`	†

where x, y and z are expressions of type *FLT*, and where xs is an expression of type `array of` *FLT*.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$max\_err\_hypot_F$	`Err_hypotenuse(`x`)`	†
$max\_err\_exp_F$	`Err_exp(`x`)`	†
$max\_err\_power_F(b,x)$	`Err_power(`b`, `x`)`	†
$max\_err\_sinh_F$	`Err_sinh(`x`)`	†
$max\_err\_tanh_F$	`Err_tanh(`x`)`	†
$big\_angle\_r_F$	`Big_radian_angle(`x`)`	†
$max\_err\_sin_F$	`Err_sin(`x`)`	†
$max\_err\_tan_F$	`Err_tan(`x`)`	†
$big\_angle\_u_F$	`Big_angle(`x`)`	†
$max\_err\_sinu_F(u)$	`Err_sin_cycle(`u`)`	†
$max\_err\_tanu_F(u)$	`Err_tan_cycle(`u`)`	†

where b, x and u are expressions of type *FLT*. Several of the parameter functions are constant for each type (and library), the argument is then used only to differentiate among the floating point types.

$hypot_F(x, y)$	`Hypotenuse(`x`, `y`)`	†
$exp_F(x)$	`Exp(`x`)`	
$expm1_F(x)$	`ExpM1(`x`)`	†
$power_{FI}(b, z)$	`Ipower(`b`, `z`)`	†
$power_F(b, y)$	`Power(`b`, `y`)`	†
$powerm1_F(b, y)$	`PowerM1(`b`, `y`)`	†
$exp2_F(x)$	`Exp2(`x`)`	†
$exp10_F(x)$	`Exp10(`x`)`	†
$ln_F(x)$	`Log(`x`)`	
$ln1p_F(x)$	`Log1P(`x`)`	†
$log_F(b, x)$	`Log(`x`, `b`)`	†
$log1p_F(b, x)$	`Log1P(`x`, `b`)`	†
$log2_F(x)$	`Log2(`x`)`	
$log10_F(x)$	`Log10(`x`)`	
$sinh_F(x)$	`Sinh(`x`)`	
$cosh_F(x)$	`Cosh(`x`)`	
$tanh_F(x)$	`Tanh(`x`)`	
$coth_F(x)$	`Coth(`x`)`	†
$sech_F(x)$	`Sech(`x`)`	†
$csch_F(x)$	`Csch(`x`)`	†
$arcsinh_F(x)$	`Arcsinh(`x`)`	†
$arccosh_F(x)$	`Arccosh(`x`)`	†
$arctanh_F(x)$	`Arctanh(`x`)`	†
$arccoth_F(x)$	`Arccoth(`x`)`	†
$arcsech_F(x)$	`Arcsech(`x`)`	†
$arccsch_F(x)$	`Arccsch(`x`)`	†
$rad\_nearest\_axis_F(x)$	`Nearest_axis(`x`)`	†
$rad\_offset\_axis_F(x)$	`Offset_axis(`x`)`	†
$rad_F(x)$	`Rad(`x`)`	†
$sin_F(x)$	`Sin(`x`)` (when in radian mode)	
$cos_F(x)$	`Cos(`x`)` (when in radian mode)	
$tan_F(x)$	`Tan(`x`)` (when in radian mode)	
$cot_F(x)$	`Cot(`x`)` (when in radian mode)	
$sec_F(x)$	`Sec(`x`)` (when in radian mode)	†
$csc_F(x)$	`Csc(`x`)` (when in radian mode)	†
$arcsin_F(x)$	`Arcsin(`x`)` (when in radian mode)	
$arccos_F(x)$	`Arccos(`x`)` (when in radian mode)	
$arctan_F(x)$	`Arctan(`x`)` (when in radian mode)	
$arccot_F(x)$	`Arccot(`x`)` (when in radian mode)	
$arcctg_F(x)$	`(Sign(`x`)*Arccot(Abs(`x`)))` (when in radian mode)	
$arcsec_F(x)$	`Arcsec(`x`)`	†
$arccsc_F(x)$	`Arccsc(`x`)`	†
$arc_F(x, y)$	`Angle(`x`, `y`)` (when in radian mode)	

$cycle\_nearest\_axis_F(u,x)$	`Nearest_axis(`x`,`u`)`	†
$cycle\_offset\_axis_F(u,x)$	`Offset_axis(`x`,`u`)`	†
$cycle_F(u,x)$	`Cycle(`u`,`x`)`	†
$sinu_F(u,x)$	`Sin(`u`,`x`)`	†
$cosu_F(u,x)$	`Cos(`u`,`x`)`	†
$tanu_F(u,x)$	`Tan(`u`,`x`)`	†
$cotu_F(u,x)$	`Cot(`u`,`x`)`	†
$secu_F(u,x)$	`Sec(`u`,`x`)`	†
$cscu_F(u,x)$	`Csc(`u`,`x`)`	†
$arcsinu_F(u,x)$	`Arcsin(`u`,`x`)`	†
$arccosu_F(u,x)$	`Arccos(`u`,`x`)`	†
$arctanu_F(u,x)$	`Arctan(`u`,`x`)`	†
$arccotu_F(u,x)$	`Arccot(`u`,`x`)`	†
$arcctgu_F(u,x)$	`(Sign(`x`)*Arccot(`u`,Abs(`x`)))`	†
$arcsecu_F(u,x)$	`Arcsec(`u`,`x`)`	†
$arccscu_F(u,x)$	`Arccsc(`u`,`x`)`	†
$arcu_F(u,x,y)$	`Angle(`u`,`x`, `y`)`	†
$deg\_nearest\_axis_F(u,x)$	`Nearest_axis(`x`)`	†
$deg\_offset\_axis_F(u,x)$	`Offset_axis(`x`)`	†
$deg_F(x)$	`Degree(`x`)`	†
$sind_F(x)$	`Sin(`x`)` (when in degree mode)	
$cosd_F(x)$	`Cos(`x`)` (when in degree mode)	
$tand_F(x)$	`Tan(`x`)` (when in degree mode)	
$cotd_F(x)$	`Cot(`x`)` (when in degree mode)	
$secd_F(x)$	`Sec(`x`)` (when in degree mode)	†
$cscd_F(x)$	`Csc(`x`)` (when in degree mode)	†
$arcsind_F(x)$	`Arcsin(`x`)` (when in degree mode)	
$arccosd_F(x)$	`Arccos(`x`)` (when in degree mode)	
$arctand_F(x)$	`Arctan(`x`)` (when in degree mode)	
$arccotd_F(x)$	`Arccot(`x`)` (when in degree mode)	
$arcctgd_F(x)$	`(Sign(`x`)*Arccot(Abs(`x`)))` (when in degree mode)	
$arcsecd_F(x)$	`Arcsec(`x`)` (when in degree mode)	†
$arccscd_F(x)$	`Arccsc(`x`)` (when in degree mode)	†
$arcd_F(x,y)$	`Angle(`x`,`y`)` (when in degree mode)	
$rad\_to\_cycle_F(x,u)$	`Rad_to_cycle(`x`, `u`)`	†
$cycle\_to\_rad_F(u,x)$	`Cycle_to_rad(`u`, `x`)`	†
$cycle\_to\_cycle_F(u,x,v)$	`Cycle_to_cycle(`u`, `x`, `v`)`	†

where b, x, y, u, and v are expressions of type *FLT*, and z is an expressions of type *INT*

Type conversions in BASIC...

$convert_{I\rightarrow I'}(x)$	$INT2(x)$

$rounding_{F\to I}(y)$	`Rounding`(y)	
$floor_{F\to I}(y)$	`Floor`(y)	
$ceiling_{F\to I}(y)$	`Ceiling`(y)	
$cvtnearest_{I\to F}(x)$		†
$cvtdown_{I\to F}(x)$		†
$cvtup_{I\to F}(x)$		†
$cvtnearest_{F\to F'}(y)$		†
$cvtdown_{F\to F'}(y)$		†
$cvtup_{F\to F'}(y)$		†
$cvtnearest_{F\to D}(y)$		†
$cvtdown_{F\to D}(y)$		†
$cvtup_{F\to D}(y)$		†
$cvtnearest_{D\to F}(z)$		†
$cvtdown_{D\to F}(z)$		†
$cvtup_{D\to F}(z)$		†

where x is an expressions of type *INT*, y is an expressions of type *FLT*, and z is an expressions of type *FXD*, where *FXD* is a fixed point type.

BASIC provides non-negative base 10 numerals for all its integer and floating point types.

C.4 C and C++

The programming language C is defined by ISO/IEC 9899:1990, *Information technology – Programming languages – C* [9]. The programming language C++ is defined by ISO/IEC , Information Technology – Programming Languages – C++.

An implementation should follow all the requirements of LIA-2 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the ISO/IEC 109672 for that operation. For each of the marked items a suggested identifier is provided.

Integer valued parameters and derived constants can be used in preprocessor expressions.

The LIA-1 datatype *Boolean* is implemented in the C datatype `int` (1 = **true** and 0 = **false**).

Every implementation of C has integral datatypes `int`, `long int`, `unsigned int`, and `unsigned long int` which conform to the LIA-1.

> NOTE – The conformity of `short` and `char` (signed or unsigned) is not relevant since values of these types are promoted to `int` (signed or unsigned) before computations are done.

C has three floating point datatypes that (can) conform to LIA-1: `float`, `double`, and `long double`.

The additional integer operations are listed below, along with the syntax used to invoke them:

$min_I(x,y)$	`imin(`x`, `y`)`	†
$max_I(x,y)$	`imax(`x`, `y`)`	†
$min\_seq_I(xs)$	`imin_arr(`xs`)`	†

$max\_seq_I(xs)$	`imax_arr(`xs`)`	†
$dim_I(x,y)$	`idim(`x`, `y`)`	†
$sqrt_I(x)$	`isqrt(`x`)`	†
$power_I(x,y)$	`ipower(`x`, `y`)`	†
$divides_I(x,y)$	`divides(`x`, `y`)`	†
$even_I(x)$	x `% 2 = 0`	
$odd_I(x)$	x `% 2 != 0`	
$gcd_I(x,y)$	`gcd(`x`, `y`)`	†
$lcm_I(x,y)$	`lcm(`x`, `y`)`	†
$gcd\_seq_I(xs)$	`gcd_arr(`xs`)`	†
$lcm\_seq_I(xs)$	`lcm_arr(`xs`)`	†
$add\_wrap_I(x,y)$	`add_wrap(`x`, `y`)`	†
$add\_ov_I(x,y)$	`add_over(`x`, `y`)`	†
$sub\_wrap_I(x,y)$	`sub_wrap(`x`, `y`)`	†
$sub\_ov_I(x,y)$	`sub_over(`x`, `y`)`	†
$mul\_wrap_I(x,y)$	`mul_wrap(`x`, `y`)`	†
$mul\_ov_I(x,y)$	`mul_over(`x`, `y`)`	†

where x and y are expressions of type INT and where xs is an expression of type $INT[]$.

The additional non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$min_F(x,y)$	`min(`x`, `y`)`	†
$max_F(x,y)$	`max(`x`, `y`)`	†
$min\_seq_F(xs)$	`min_arr(`xs`)`	†
$max\_seq_F(xs)$	`max_arr(`xs`)`	†
$rounding_F(x)$	`round(`x`)`	†
$floor_F(x)$	`ffloorf(`x`)`, `ffloor(`x`)`, `ffloorl(`x`)`	†
$ceiling_F(x)$	`ceiling(`x`)`	†
$dim_F(x,y)$	`dim(`x`, `y`)`	†
$add3_F(x,y,z)$	`add(`x`, `y`, `z`)`	†
$sum_F(xs)$	`sum(`xs`)`	†
$dprod_{F\rightarrow F'}(x,y)$	`????(`x`, `y`)`	†
$mul\_add_F(x,y,z)$	`mul_add(`x`, `y`, `z`)`	†
$irem_F(x,y)$	`remainder(`x`, `y`)`	†
$sqrt_F(x)$	`sqrtf(`x`)`, `sqrt(`x`)`, `sqrtl(`x`)`	†
$rsqrt_F(x)$	`rsqrt(`x`)`	†
$add\_lo_F(x,y)$	`add_low(`x`, `y`)`	†
$sub\_lo_F(x,y)$	`sub_low(`x`, `y`)`	†
$mul\_lo_F(x,y)$	`mul_low(`x`, `y`)`	†
$div\_rest_F(x,y)$	`div_rest(`x`, `y`)`	†
$sqrt\_rest_F(x)$	`sqrt_rest(`x`)`	†

where x, y and z are expressions of type FLT, and where xs is an expression of type $FLT[]$.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$max\_err\_hypot_F$	`Err_hypotenuse(`x`)`	†
$max\_err\_exp_F$	`Err_exp(`x`)`	†
$max\_err\_power_F(b,x)$	`Err_power(`b`, `x`)`	†
$max\_err\_sinh_F$	`Err_sinh(`x`)`	†
$max\_err\_tanh_F$	`Err_tanh(`x`)`	†
$big\_radian\_angle_F$	`Big_radian_angle(`x`)`	†
$max\_err\_sin_F$	`Err_sin(`x`)`	†
$max\_err\_tan_F$	`Err_tan(`x`)`	†
big_angle_F	`Big_angle(`x`)`	†
$max\_err\_sinu_F(u)$	`Err_sin_cycle(`u`)`	†
$max\_err\_tanu_F(u)$	`Err_tan_cycle(`u`)`	†

where b, x and u are expressions of type *FLT*. Several of the parameter functions are constant for each type (and library), the argument is then used only to differentiate among the floating point types.

$hypot_F(x,y)$	`hypotenuse(`x`, `y`)`	†
$exp_F(x)$	`exp(`x`)`	
$expm1_F(x)$	`expm1(`x`)`	
$power_{FI}(b,z)$	`poweri(`b`, `z`)`	†
$power_F(b,y)$	`power(`b`, `y`)`	†
$powerm1_F(b,y)$	`powerm1(`b`, `y`)`	†
$exp2_F(x)$	`exp2(`x`)`	†
$exp10_F(x)$	`exp10(`x`)`	†
$ln_F(x)$	`ln(`x`)`	
$ln1p_F(x)$	`ln1p(`x`)`	
$log_F(b,x)$	`log(`b`, `x`)`	†
$log1p_F(b,x)$	`log1p(`b`, `x`)`	†
$log2_F(x)$	`log2(`x`)`	†
$log10_F(x)$	`log10(`x`)`	
$sinh_F(x)$	`sinh(`x`)`	
$cosh_F(x)$	`cosh(`x`)`	
$tanh_F(x)$	`tanh(`x`)`	
$coth_F(x)$	`coth(`x`)`	†
$sech_F(x)$	`sech(`x`)`	†
$csch_F(x)$	`csch(`x`)`	†
$arcsinh_F(x)$	`asinh(`x`)`	
$arccosh_F(x)$	`acosh(`x`)`	
$arctanh_F(x)$	`atanh(`x`)`	
$arccoth_F(x)$	`acoth(`x`)`	†

$arcsech_F(x)$	asech(x)	†
$arccsch_F(x)$	acsch(x)	†
$rad\_nearest\_axis_F(x)$	nearest_axis(x)	†
$rad\_offset\_axis_F(x)$	offset_axis(x)	†
$rad_F(x)$	radian(x)	†
$sin_F(x)$	sin(x)	
$cos_F(x)$	cos(x)	
$tan_F(x)$	tan(x)	
$cot_F(x)$	cot(x)	†
$sec_F(x)$	sec(x)	†
$csc_F(x)$	csc(x)	†
$arcsin_F(x)$	asin(x)	
$arccos_F(x)$	acos(x)	
$arctan_F(x)$	atan(x)	
$arccot_F(x)$	acot(x)	†
$arcctg_F(x)$	(sign(x)*acot(abs(x)))	†
$arcsec_F(x)$	asec(x)	†
$arccsc_F(x)$	acsc(x)	†
$arc_F(x,y)$	angle(x, y)	†
$cycle\_nearest\_axis_F(u,x)$	nearest_axisu(u, x)	†
$cycle\_offset\_axis_F(u,x)$	offset_axisu(u, x)	†
$cycle_F(u,x)$	cycle(u, x)	†
$sinu_F(u,x)$	sinu(u, x)	†
$cosu_F(u,x)$	cosu(u, x)	†
$tanu_F(u,x)$	tanu(u, x)	†
$cotu_F(u,x)$	cotu(u, x)	†
$secu_F(u,x)$	secu(u, x)	†
$cscu_F(u,x)$	cscu(u, x)	†
$arcsinu_F(u,x)$	asinu(u, x)	†
$arccosu_F(u,x)$	acosu(u, x)	†
$arctanu_F(u,x)$	atanu(u, x)	†
$arccotu_F(u,x)$	acotu(u, x)	†
$arcctgu_F(u,x)$	(sign(x)*acotu(u, abs(x)))	†
$arcsecu_F(u,x)$	asecu(u, x)	†
$arccscu_F(u,x)$	acscu(u, x)	†
$arcu_F(u,x,y)$	angleu(u, x, y)	†
$deg\_nearest\_axis_F(u,x)$	nearest_axisu(360.0, x)	†
$deg\_offset\_axis_F(u,x)$	offset_axisu(360.0, x)	†
$deg_F(x)$	cycle(360.0, x)	†
$sind_F(x)$	sinu(360.0, x)	†
$cosd_F(x)$	cosu(360.0, x)	†
$tand_F(x)$	tanu(360.0, x)	†
$cotd_F(x)$	cotu(360.0, x)	†

$secd_F(x)$	`secu(360.0,` x`)`	†
$cscd_F(x)$	`cscu(360.0,` x`)`	†
$arcsind_F(x)$	`asinu(360.0,` x`)`	†
$arccosd_F(x)$	`acosu(360.0,` x`)`	†
$arctand_F(x)$	`atanu(360.0,` x`)`	†
$arccotd_F(x)$	`acotu(360.0,` x`)`	†
$arcctgd_F(x)$	`(sign(`x`)*acotu(360.0, abs(`x`)))`	†
$arcsecd_F(x)$	`asecu(360.0,` x`)`	†
$arccscd_F(x)$	`acscu(360.0,` x`)`	†
$arcd_F(x,y)$	`angleu(360.0,` x`,`y`)`	†
$rad\_to\_cycle_F(x,u)$	`radian_to_cycle(`x`,` u`)`	†
$cycle\_to\_rad_F(u,x)$	`cycle_to_radian(`u`,` x`)`	†
$cycle\_to\_cycle_F(u,x,v)$	`cycle_to_cycle(`u`,` x`,` v`)`	†

where b, x, y, u, and v are expressions of type FLT, and z is an expressions of type INT

....

$convert_{I\to I'}(x)$		†
$rounding_{F\to I}(y)$	`rounding(`y`)`	†
$floor_{F\to I}(y)$	`floor(`y`)`	
$ceiling_{F\to I}(y)$	`ceil(`y`)`	
$cvtnearest_{I\to F}(x)$		†
$cvtdown_{I\to F}(x)$		†
$cvtup_{I\to F}(x)$		†
$cvtnearest_{F\to F'}(y)$		†
$cvtdown_{F\to F'}(y)$		†
$cvtup_{F\to F'}(y)$		†
$cvtnearest_{F\to D}(y)$		†
$cvtdown_{F\to D}(y)$		†
$cvtup_{F\to D}(y)$		†
$cvtnearest_{D\to F}(z)$		†
$cvtdown_{D\to F}(z)$		†
$cvtup_{D\to F}(z)$		†

where x is an expressions of type INT, y is an expressions of type FLT, and z is an expressions of type FXD, where FXD is a fixed point type.

C provides non-negative numerals .....

C.5 Fortran

The programming language Fortran is defined by ISO/IEC 1539:1991, *Information technology – Programming languages – FORTRAN* [3].

An implementation should follow all the requirements of LIA-2 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the LIA-2 for that operation. For each of the marked items a suggested identifier is provided.

The Fortran datatype `LOGICAL` corresponds to the LIA-1 datatype *Boolean*.

Every implementation of Fortran has one integer data type, denoted as `INTEGER`, and two floating point data type denoted as `REAL` (single precision) and `DOUBLE PRECISION`.

An implementation is permitted to offer additional `INTEGER` types with a different range and additional `REAL` types with different precision or range, parameterized with the `KIND` parameter.

The additional integer operations are listed below, along with the syntax used to invoke them:

$min_I(x,y)$	`min0(`x`, `y`)`	
$max_I(x,y)$	`max0(`x`, `y`)`	
$min\_seq_I(xs)$	`min0(`xs`[1], `xs`[2], ..., `xs`[`n`])`	
$max\_seq_I(xs)$	`max0(`xs`[1], `xs`[2], ..., `xs`[`n`])`	
$dim_I(x,y)$	`dim(`x`, `y`)`	
$sqrt_I(x)$	`isqrt(`x`)`	†
$power_I(x,y)$	x `**` y	
$divides_I(x,y)$	`divides(`x`, `y`)`	†
$even_I(x)$	`even(`x`)`	†
$odd_I(x)$	`odd(`x`)`	†
$gcd_I(x,y)$	`gcd(`x`, `y`)`	†
$lcm_I(x,y)$	`lcm(`x`, `y`)`	†
$gcd\_seq_I(xs)$	`gcd(`xs`)`	†
$lcm\_seq_I(xs)$	`lcm(`xs`)`	†
$add\_wrap_I(x,y)$	`add_wrap(`x`, `y`)`	†
$add\_ov_I(x,y)$	`add_over(`x`, `y`)`	†
$sub\_wrap_I(x,y)$	`sub_wrap(`x`, `y`)`	†
$sub\_ov_I(x,y)$	`sub_over(`x`, `y`)`	†
$mul\_wrap_I(x,y)$	`mul_wrap(`x`, `y`)`	†
$mul\_ov_I(x,y)$	`mul_over(`x`, `y`)`	†

where x and y are expressions of type *INT* and where xs is an expression of type `ARRAY OF` *INT*.

The additional non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$min_F(x,y)$	`amin1(`x`, `y`)`
$max_F(x,y)$	`amax1(`x`, `y`)`
$min\_seq_F(xs)$	`amin1(`xs`[1], `xs`[2], ..., `xs`[`n`])`
$max\_seq_F(xs)$	`amax1(`xs`[1], `xs`[2], ..., `xs`[`n`])`
$floor_F(x)$	`floor(`x`)`
$rounding_F(x)$	

$ceiling_F(x)$	`ceil(`x`)`	
$dim_F(x,y)$	`dim(`x`, `y`)`	
$add3_F(x,y,z)$	`add(`x`, `y`, `z`)`	†
$sum_F(xs)$	`sum(`xs`)`	†
$dprod_{F\to F'}(x,y)$	`????(`x`, `y`)`	†
$mul\_add_F(x,y,z)$	`mul_add(`x`, `y`, `z`)`	†
$irem_F(x,y)$	`remainder(`x`, `y`)`	†
$sqrt_F(x)$	`sqrt(`x`)`	
$rsqrt_F(x)$	`rsqrt(`x`)`	†
$add\_lo_F(x,y)$	`add_low(`x`, `y`)`	†
$sub\_lo_F(x,y)$	`sub_low(`x`, `y`)`	†
$mul\_lo_F(x,y)$	`mul_low(`x`, `y`)`	†
$div\_rest_F(x,y)$	`div_rest(`x`, `y`)`	†
$sqrt\_rest_F(x)$	`sqrt_rest(`x`)`	†

where x, y and z are expressions of type *FLT*, and where xs is an expression of type `array of` *FLT*.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$max\_err\_hypot_F$	`err_hypotenuse(`x`)`	†
$max\_err\_exp_F$	`err_exp(`x`)`	†
$max\_err\_power_F(b,x)$	`err_power(`b`, `x`)`	†
$max\_err\_sinh_F$	`err_sinh(`x`)`	†
$max\_err\_tanh_F$	`err_tanh(`x`)`	†
$big\_angle\_r_F$	`big_radian_angle(`x`)`	†
$max\_err\_sin_F$	`err_sin(`x`)`	†
$max\_err\_tan_F$	`err_tan(`x`)`	†
$big\_angle\_u_F$	`big_angle(`x`)`	†
$max\_err\_sinu_F(u)$	`err_sin_cycle(`u`)`	†
$max\_err\_tanu_F(u)$	`err_tan_cycle(`u`)`	†

where b, x and u are expressions of type *FLT*. Several of the parameter functions are constant for each type (and library), the argument is then used only to differentiate among the floating point types.

$hypot_F(x,y)$	`hypotenuse(`x`, `y`)`	†
$power_{FI}(b,z)$	b `**` z	
$power_F(b,y)$	b `**` y	
$powerm1_F(b,y)$	`powerm1(`b`, `y`)`	†
$exp_F(x)$	`exp(`x`)`	
$expm1_F(x)$	`expm1(`x`)`	†
$exp2_F(x)$	`exp2(`x`)`	†
$exp10_F(x)$	`exp10(`x`)`	†

$log_F(b,x)$	`logbase(`b`, `x`)`	†
$log1p_F(b,x)$	`logbase1p(`b`, `x`)`	†
$ln_F(x)$	`log(`x`)`	
$ln1p_F(x)$	`log1p(`x`)`	†
$log2_F(x)$	`log2(`x`)`	
$log10_F(x)$	`log10(`x`)`	
$sinh_F(x)$	`sinh(`x`)`	
$cosh_F(x)$	`cosh(`x`)`	
$tanh_F(x)$	`tanh(`x`)`	
$coth_F(x)$	`coth(`x`)`	†
$sech_F(x)$	`sech(`x`)`	†
$csch_F(x)$	`csch(`x`)`	†
$arcsinh_F(x)$	`asinh(`x`)`	
$arccosh_F(x)$	`acosh(`x`)`	
$arctanh_F(x)$	`atanh(`x`)`	
$arccoth_F(x)$	`acoth(`x`)`	†
$arcsech_F(x)$	`asech(`x`)`	†
$arccsch_F(x)$	`acsch(`x`)`	†
$rad\_nearest\_axis_F(x)$	`rad_nearest_axis(`x`)`	†
$rad\_offset\_axis_F(x)$	`rad_offset_axis(`x`)`	†
$rad_F(x)$	`rad(`x`)`	†
$sin_F(x)$	`sin(`x`)`	
$cos_F(x)$	`cos(`x`)`	
$tan_F(x)$	`tan(`x`)`	
$cot_F(x)$	`cot(`x`)`	
$sec_F(x)$	`sec(`x`)`	†
$csc_F(x)$	`csc(`x`)`	†
$arcsin_F(x)$	`asin(`x`)`	
$arccos_F(x)$	`acos(`x`)`	
$arctan_F(x)$	`atan(`x`)`	
$arccot_F(x)$	`acot(`x`)`	†
$arcctg_F(x)$	`(sign(`x`)*acot(abs(`x`)))`	†
$arcsec_F(x)$	`asec(`x`)`	†
$arccsc_F(x)$	`acsc(`x`)`	†
$arc_F(x,y)$	`atan2(`y`, `x`)`	
$cycle\_nearest\_axis_F(u,x)$	`cycle_nearest_axis(`u`,`x`)`	†
$cycle\_offset\_axis_F(u,x)$	`cycle_offset_axis(`u`,`x`)`	†
$cycle_F(u,x)$	`cycle(`u`,`x`)`	†
$sinu_F(u,x)$	`sinu(`u`,`x`)`	†
$cosu_F(u,x)$	`cosu(`u`,`x`)`	†
$tanu_F(u,x)$	`tanu(`u`,`x`)`	†
$cotu_F(u,x)$	`cotu(`u`,`x`)`	†
$secu_F(u,x)$	`secu(`u`,`x`)`	†

$cscu_F(u,x)$	`cscu(`u`,`x`)`	†
$arcsinu_F(u,x)$	`asinu(`u`,`x`)`	†
$arccosu_F(u,x)$	`acosu(`u`,`x`)`	†
$arctanu_F(u,x)$	`atanu(`u`,`x`)`	†
$arccotu_F(u,x)$	`acotu(`u`,`x`)`	†
$arcctgu_F(u,x)$	`(sign(`x`)*acotu(abs(`x`),`u`))`	
$arcsecu_F(u,x)$	`asecu(`u`,`x`)`	†
$arccscu_F(u,x)$	`acscu(`u`,`x`)`	†
$arcu_F(u,x,y)$	`atan2u(`u`,`x`,`y`)`	†
$deg\_nearest\_axis_F(u,x)$	`degree_nearest_axis(`x`)`	†
$deg\_offset\_axis_F(u,x)$	`degree_offset_axis(`x`)`	†
$deg_F(x)$	`degrees(`x`)`	†
$sind_F(x)$	`sind(`x`)`	†
$cosd_F(x)$	`cosd(`x`)`	†
$tand_F(x)$	`tand(`x`)`	†
$cotd_F(x)$	`cotd(`x`)`	†
$secd_F(x)$	`secd(`x`)`	†
$cscd_F(x)$	`cscd(`x`)`	†
$arcsind_F(x)$	`asind(`x`)`	†
$arccosd_F(x)$	`acosd(`x`)`	†
$arctand_F(x)$	`atand(`x`)`	†
$arccotd_F(x)$	`acotd(`x`)`	†
$arcctgd_F(x)$	`(sign(`x`)*acotd(abs(`x`)))`	†
$arcsecd_F(x)$	`asecd(`x`)`	†
$arccscd_F(x)$	`acscd(`x`)`	†
$arcd_F(x,y)$	`atan2d(`y`,`x`)`	†
$rad\_to\_cycle_F(x,u)$	`rad_to_cycle(`x`, `u`)`	†
$cycle\_to\_rad_F(u,x)$	`cycle_to_rad(`u`, `x`)`	†
$cycle\_to\_cycle_F(u,x,v)$	`cycle_to_cycle(`u`, `x`, `v`)`	†

where b, x, y, u, and v are expressions of type FLT, and z is an expressions of type INT

Type conversions in Fortran...

$convert_{I\to I'}(x)$	$INT2(x)$	
$rounding_{F\to I}(y)$	`nint(`y`))`	
$floor_{F\to I}(y)$	`floor(`y`)`	
$ceiling_{F\to I}(y)$	`ceiling(`y`)`	
$cvtnearest_{I\to F}(x)$		†
$cvtdown_{I\to F}(x)$		†
$cvtup_{I\to F}(x)$		†
$cvtnearest_{F\to F'}(y)$		†

$cvtdown_{F \to F'}(y)$	†
$cvtup_{F \to F'}(y)$	†
$cvtnearest_{F \to D}(y)$	†
$cvtdown_{F \to D}(y)$	†
$cvtup_{F \to D}(y)$	†
$cvtnearest_{D \to F}(z)$	†
$cvtdown_{D \to F}(z)$	†
$cvtup_{D \to F}(z)$	†

where x is an expressions of type *INT*, y is an expressions of type *FLT*, and z is an expressions of type *FXD*, where *FXD* is a fixed point type.

Fortran provides non-negative numerals for all its integer and floating point types.

C.6 Java

The programming language Java is defined by SUN Microsystems...

An implementation should follow all the requirements of LIA-2 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the LIA-2 for that operation. For each of the marked items a suggested identifier is provided.

The Java datatype `Boolean` corresponds to the LIA-1 datatype *Boolean*.

Every implementation of Java has integral types `int`, `long int`, `unsigned int`, and `unsigned long int` which (can) conform to the LIA-1.

Java has three floating point types that (can) conform to LIA-1: `float`, `double`, and `long double`.

The additional integer operations are listed below, along with the syntax used to invoke them:

$min_I(x,y)$	`imin(`x`, `y`)`	†
$max_I(x,y)$	`imax(`x`, `y`)`	†
$min\_seq_I(xs)$	`imin_arr(`xs`)`	†
$max\_seq_I(xs)$	`imax_arr(`xs`)`	†
$dim_I(x,y)$	`idim(`x`, `y`)`	†
$sqrt_I(x)$	`isqrt(`x`)`	†
$power_I(x,y)$	`ipower(`x`, `y`)`	†
$divides_I(x,y)$	`divides(`x`, `y`)`	†
$even_I(x)$	x `% 2 = 0`	
$odd_I(x)$	x `% 2 != 0`	
$gcd_I(x,y)$	`gcd(`x`, `y`)`	†
$lcm_I(x,y)$	`lcm(`x`, `y`)`	†
$gcd\_seq_I(xs)$	`gcd_arr(`xs`)`	†
$lcm\_seq_I(xs)$	`lcm_arr(`xs`)`	†
$add\_wrap_I(x,y)$	`add_wrap(`x`, `y`)`	†
$add\_ov_I(x,y)$	`add_over(`x`, `y`)`	†

$sub\_wrap_I(x,y)$	`sub_wrap(`x`, `y`)`	†
$sub\_ov_I(x,y)$	`sub_over(`x`, `y`)`	†
$mul\_wrap_I(x,y)$	`mul_wrap(`x`, `y`)`	†
$mul\_ov_I(x,y)$	`mul_over(`x`, `y`)`	†

where x and y are expressions of type *INT* and where xs is an expression of type `array of` *INT*.

The additional non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$min_F(x,y)$	`min(`x`, `y`)`	†
$max_F(x,y)$	`max(`x`, `y`)`	†
$min\_seq_F(xs)$	`min_arr(`xs`)`	†
$max\_seq_F(xs)$	`max_arr(`xs`)`	†
$floor_F(x)$	`ffloor(`x`)`	
$rounding_F(x)$	`round(`x`)`	†
$ceiling_F(x)$	`ceiling(`x`)`	†
$dim_F(x,y)$	`dim(`x`, `y`)`	†
$add3_F(x,y,z)$	`add(`x`, `y`, `z`)`	†
$sum_F(xs)$	`sum(`xs`)`	†
$dprod_{F\to F'}(x,y)$	`????(`x`, `y`)`	†
$mul\_add_F(x,y,z)$	`mul_add(`x`, `y`, `z`)`	†
$irem_F(x,y)$	`remainder(`x`, `y`)`	†
$sqrt_F(x)$	`sqrt(`x`)`	
$rsqrt_F(x)$	`rsqrt(`x`)`	†
$add\_lo_F(x,y)$	`add_low(`x`, `y`)`	†
$sub\_lo_F(x,y)$	`sub_low(`x`, `y`)`	†
$mul\_lo_F(x,y)$	`mul_low(`x`, `y`)`	†
$div\_rest_F(x,y)$	`div_rest(`x`, `y`)`	†
$sqrt\_rest_F(x)$	`sqrt_rest(`x`)`	†

where x, y and z are expressions of type *FLT*, and where xs is an expression of type `array of` *FLT*.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$max\_err\_hypot_F$	`Err_hypotenuse(`x`)`	†
$max\_err\_exp_F$	`Err_exp(`x`)`	†
$max\_err\_power_F(b,x)$	`Err_power(`b`, `x`)`	†
$max\_err\_sinh_F$	`Err_sinh(`x`)`	†
$max\_err\_tanh_F$	`Err_tanh(`x`)`	†
$big\_radian\_angle_F$	`Big_radian_angle(`x`)`	†
$max\_err\_sin_F$	`Err_sin(`x`)`	†
$max\_err\_tan_F$	`Err_tan(`x`)`	†
big_angle_F	`Big_angle(`x`)`	†
$max\_err\_sinu_F(u)$	`Err_sin_cycle(`u`)`	†

$max\_err\_tanu_F(u)$	`Err_tan_cycle(`u`)`	†

where b, x and u are expressions of type *FLT*. Several of the parameter functions are constant for each type (and library), the argument is then used only to differentiate among the floating point types.

$hypot_F(x, y)$	`hypotenuse(`x`, `y`)`	†
$exp_F(x)$	`exp(`x`)`	
$expm1_F(x)$	`expm1(`x`)`	
$power_{FI}(b, z)$	`poweri(`b`, `z`)`	†
$power_F(b, y)$	`power(`b`, `y`)`	†
$powerm1_F(b, y)$	`powerm1(`b`, `y`)`	†
$exp2_F(x)$	`exp2(`x`)`	†
$exp10_F(x)$	`exp10(`x`)`	†
$ln_F(x)$	`ln(`x`)`	
$ln1p_F(x)$	`ln1p(`x`)`	
$log_F(b, x)$	`log(`b`, `x`)`	†
$log1p_F(b, x)$	`log1p(`b`, `x`)`	†
$log2_F(x)$	`log2(`x`)`	†
$log10_F(x)$	`log10(`x`)`	
$sinh_F(x)$	`sinh(`x`)`	
$cosh_F(x)$	`cosh(`x`)`	
$tanh_F(x)$	`tanh(`x`)`	
$coth_F(x)$	`coth(`x`)`	†
$sech_F(x)$	`sech(`x`)`	†
$csch_F(x)$	`csch(`x`)`	†
$arcsinh_F(x)$	`asinh(`x`)`	
$arccosh_F(x)$	`acosh(`x`)`	
$arctanh_F(x)$	`atanh(`x`)`	
$arccoth_F(x)$	`acoth(`x`)`	†
$arcsech_F(x)$	`asech(`x`)`	†
$arccsch_F(x)$	`acsch(`x`)`	†
$rad\_nearest\_axis_F(x)$	`nearest_axis(`x`)`	†
$rad\_offset\_axis_F(x)$	`offset_axis(`x`)`	†
$rad_F(x)$	`radian(`x`)`	†
$sin_F(x)$	`sin(`x`)`	
$cos_F(x)$	`cos(`x`)`	
$tan_F(x)$	`tan(`x`)`	
$cot_F(x)$	`cot(`x`)`	†
$sec_F(x)$	`sec(`x`)`	†
$csc_F(x)$	`csc(`x`)`	†
$arcsin_F(x)$	`asin(`x`)`	
$arccos_F(x)$	`acos(`x`)`	
$arctan_F(x)$	`atan(`x`)`	

$arccot_F(x)$	`acot(`x`)`	†
$arcctg_F(x)$	`(sign(`x`)*acot(abs(`x`)))`	†
$arcsec_F(x)$	`asec(`x`)`	†
$arccsc_F(x)$	`acsc(`x`)`	†
$arc_F(x,y)$	`angle(`x`, `y`)`	†
$cycle\_nearest\_axis_F(u,x)$	`nearest_axisu(`u`, `x`)`	†
$cycle\_offset\_axis_F(u,x)$	`offset_axisu(`u`, `x`)`	†
$cycle_F(u,x)$	`Cycle(`u`, `x`)`	†
$sinu_F(u,x)$	`sinu(`u`, `x`)`	†
$cosu_F(u,x)$	`cosu(`u`, `x`)`	†
$tanu_F(u,x)$	`tanu(`u`, `x`)`	†
$cotu_F(u,x)$	`cotu(`u`, `x`)`	†
$secu_F(u,x)$	`secu(`u`, `x`)`	†
$cscu_F(u,x)$	`cscu(`u`, `x`)`	†
$arcsinu_F(u,x)$	`asinu(`u`, `x`)`	†
$arccosu_F(u,x)$	`acosu(`u`, `x`)`	†
$arctanu_F(u,x)$	`atanu(`u`, `x`)`	†
$arccotu_F(u,x)$	`acotu(`u`, `x`)`	†
$arcctgu_F(u,x)$	`(sign(`x`)*acotu(`u`, abs(`x`)))`	†
$arcsecu_F(u,x)$	`asecu(`u`, `x`)`	†
$arccscu_F(u,x)$	`acscu(`u`, `x`)`	†
$arcu_F(u,x,y)$	`angleu(`u`, `x`, `y`)`	†
$deg\_nearest\_axis_F(u,x)$	`nearest_axisu(360.0, `x`)`	†
$deg\_offset\_axis_F(u,x)$	`offset_axisu(360.0, `x`)`	†
$deg_F(x)$	`cycle(360.0, `x`)`	†
$sind_F(x)$	`sinu(360.0, `x`)`	†
$cosd_F(x)$	`cosu(360.0, `x`)`	†
$tand_F(x)$	`tanu(360.0, `x`)`	†
$cotd_F(x)$	`cotu(360.0, `x`)`	†
$secd_F(x)$	`secu(360.0, `x`)`	†
$cscd_F(x)$	`cscu(360.0, `x`)`	†
$arcsind_F(x)$	`asin(360.0, `x`)`	†
$arccosd_F(x)$	`acos(360.0, `x`)`	†
$arctand_F(x)$	`atan(360.0, `x`)`	†
$arccotd_F(x)$	`acot(360.0, `x`)`	†
$arcctgd_F(x)$	`(sign(`x`)*acot(360.0, abs(`x`)))`	†
$arcsecd_F(x)$	`asec(360.0, `x`)`	†
$arccscd_F(x)$	`acsc(360.0, `x`)`	†
$arcd_F(x,y)$	`angle(360.0, `x`,`y`)`	†
$rad\_to\_cycle_F(x,u)$	`radian_to_cycle(`x`, `u`)`	†
$cycle\_to\_rad_F(u,x)$	`cycle_to_radian(`u`, `x`)`	†
$cycle\_to\_cycle_F(u,x,v)$	`cycle_to_cycle(`u`, `x`, `v`)`	†

where b, x, y, u, and v are expressions of type *FLT*, and z is an expressions of type *INT*

....

$convert_{I \to I'}(x)$		†
$floor_{F \to I}(y)$	`floor`(y)	
$rounding_{F \to I}(y)$	`round`(y)	†
$ceiling_{F \to I}(y)$	`ceil`(y)	
$cvtdown_{I \to F}(x)$		†
$cvtnearest_{I \to F}(x)$		†
$cvtup_{I \to F}(x)$		†
$cvtdown_{F \to F'}(y)$		†
$cvtnearest_{F \to F'}(y)$		†
$cvtup_{F \to F'}(y)$		†
$cvtdown_{F \to D}(y)$		†
$cvtnearest_{F \to D}(y)$		†
$cvtup_{F \to D}(y)$		†
$cvtdown_{D \to F}(z)$		†
$cvtnearest_{D \to F}(z)$		†
$cvtup_{D \to F}(z)$		†

where x is an expressions of type *INT*, y is an expressions of type *FLT*, and z is an expressions of type *FXD*, where *FXD* is a fixed point type.

Java provides non-negative numerals .....

C.7 ISLisp and Common Lisp

The programming language ISLisp is defined by ISO/IEC CD 13816.2, Information Technology – Programming Languages – Lisp .

The programming language Common Lisp is under development by ANSI X3J13.The standard will be based on the definition contained in *Common Lisp the Language.*

An implementation should follow all the requirements of LIA-2 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the LIA-2 for that operation. For each of the marked items a suggested identifier is provided.

Common Lisp does not have a single datatype that corresponds to the LIA-1 datatype *Boolean*. Rather, `NIL` corresponds to **false** and `T` corresponds to **true**.

Every implementation of Common Lisp and of ISLisp has one unbounded integer datatype. Any mathematical integer is assumed to have a representation as a Common Lisp or ISLisp data object, subject only to total memory limitations.

Common Lisp has four floating point types: `short-float`, `single-float`, `double-float`, and `long-float`. Not all of these floating point types must be distinct.

The additional integer operations are listed below, along with the syntax used to invoke them:

$min_I(x,y)$	(min x y)	
$max_I(x,y)$	(max x y)	
$min\_seq_I(xs)$	(min.xs) or (min x_1 x_2 ... x_n)	
$max\_seq_I(xs)$	(max.xs) or (max x_1 x_2 ... x_n)	
$dim_I(x,y)$	(dim x y)	†
$sqrt_I(x)$	(isqrt x)	†
$power_I(x,y)$	(power x y)	†
$divides_I(x,y)$	(dividesp x y)	†
$even_I(x)$	(evenp x)	
$odd_I(x)$	(oddp x)	
$gcd_I(x,y)$	(gcd x y)	
$lcm_I(x,y)$	(lcm x y)	
$gcd\_seq_I(xs)$	(gcd.xs) or (gcd x_1 x_2 ... x_n)	
$lcm\_seq_I(xs)$	(lcm.xs) or (lcm x_1 x_2 ... x_n)	
$add\_wrap_I(x,y)$	(add_wrap x y)	†
$add\_ov_I(x,y)$	(add_over x y)	†
$sub\_wrap_I(x,y)$	(sub_wrap x y)	†
$sub\_ov_I(x,y)$	(sub_over x y)	†
$mul\_wrap_I(x,y)$	(mul_wrap x y)	†
$mul\_ov_I(x,y)$	(mul_over x y)	†

where x and y are expressions of type *INT* and where xs is an expression of type list of *INT*.

The additional non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$min_F(x,y)$	(min x y)	
$max_F(x,y)$	(max x y)	
$min\_seq_F(xs)$	(min.xs) or (min x_1 x_2 ... x_n)	
$max\_seq_F(xs)$	(max.xs) or (max x_1 x_2 ... x_n)	
$floor_F(x)$	(ffloor x)	
$rounding_F(x)$	(fround x)	
$ceiling_F(x)$	(fceiling x)	
$dim_F(x,y)$	(dim x y)	†
$add3_F(x,y,z)$	(add x y z)	†
$sum_F(xs)$	(sum xs)	†
$dprod_{F\to F'}(x,y)$	(prod x y)	†
$mul\_add_F(x,y,z)$	(mul_add x y z)	†
$irem_F(x,y)$	(remainder x y)	†
$sqrt_F(x)$	(sqrt x)	
$rsqrt_F(x)$	(rsqrt x)	†
$add\_lo_F(x,y)$	(add_low x y)	†
$sub\_lo_F(x,y)$	(sub_low x y)	†
$mul\_lo_F(x,y)$	(mul_low x y)	†
$div\_rest_F(x,y)$	(div_rest x y)	†

$sqrt\_rest_F(x)$	(sqrt_rest x)	†

where x, y and z are data objects of the same floating point type, and where xs is an data objects that are lists of data objects of the same floating point type.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$max\_err\_hypot_F$	(err_hypotenuse x)	†
$max\_err\_exp_F$	(err_exp x)	†
$max\_err\_power_F(b,x)$	(err_power b x)	†
$max\_err\_sinh_F$	(err_sinh x)	†
$max\_err\_tanh_F$	(err_tanh x)	†
$big\_radian\_angle_F$	(big_radian_angle x)	†
$max\_err\_sin_F$	(err_sin x)	†
$max\_err\_tan_F$	(err_tan x)	†
big_angle_F	(big_angle x)	†
$max\_err\_sinu_F(u)$	(err_sin_cycle u)	†
$max\_err\_tanu_F(u)$	(err_tan_cycle u)	†

where b, x and u are expressions of type *FLT*. Several of the parameter functions are constant for each type (and library), the argument is then used only to differentiate among the floating point types.

$hypot_F(x,y)$	(hypotenuse x y)	†
$exp_F(x)$	(expt x)	
$expm1_F(x)$	(expm1 x)	†
$power_{FI}(b,z)$	(expt b z)	
$power_F(b,y)$	(expt b y)	
$powerm1_F(b,y)$	(expm1 b y)	†
$exp2_F(x)$	(exp2 x)	†
$exp10_F(x)$	(exp10 x)	†
$ln_F(x)$	(log x)	
$ln1p_F(x)$	(log1p x)	†
$log_F(b,x)$	(log b x)	
$log1p_F(b,x)$	(log1p b x)	†
$log2_F(x)$	(log2 x)	†
$log10_F(x)$	(log10 x)	†
$sinh_F(x)$	(sinh x)	
$cosh_F(x)$	(cosh x)	
$tanh_F(x)$	(tanh x)	
$coth_F(x)$	(coth x)	†
$sech_F(x)$	(sech x)	†
$csch_F(x)$	(csch x)	†

$arcsinh_F(x)$	(asinh x)	
$arccosh_F(x)$	(acosh x)	
$arctanh_F(x)$	(atanh x)	
$arccoth_F(x)$	(acoth x)	†
$arcsech_F(x)$	(asech x)	†
$arccsch_F(x)$	(acsch x)	†
$rad\_nearest\_axis_F(x)$	(rad_nearest_axis x)	†
$rad\_offset\_axis_F(x)$	(rad_offset_axis x)	†
$rad_F(x)$	(radians x)	†
$sin_F(x)$	(sin x)	
$cos_F(x)$	(cos x)	
$tan_F(x)$	(tan x)	
$cot_F(x)$	(cot x)	†
$sec_F(x)$	(sec x)	†
$csc_F(x)$	(csc x)	†
$arcsin_F(x)$	(asin x)	
$arccos_F(x)$	(acos x)	
$arctan_F(x)$	(atan x)	
$arccot_F(x)$	(acot x)	†
$arcctg_F(x)$	(* (sign x) (acot (abs x)))	†
$arcsec_F(x)$	(asec x)	†
$arccsc_F(x)$	(acsc x)	†
$arc_F(x,y)$	(atan2 y x)	
$cycle\_nearest\_axis_F(u,x)$	(cycle_nearest_axis u x)	†
$cycle\_offset\_axis_F(u,x)$	(cycle_offset_axis u x)	†
$cycle_F(u,x)$	(cycle u x)	†
$sinu_F(u,x)$	(sinU u x)	†
$cosu_F(u,x)$	(cosU u x)	†
$tanu_F(u,x)$	(tanU u x)	†
$cotu_F(u,x)$	(cotU u x)	†
$secu_F(u,x)$	(secU u x)	†
$cscu_F(u,x)$	(cscU u x)	†
$arcsinu_F(u,x)$	(asinU u x	†
$arccosu_F(u,x)$	(acosU u x	†
$arctanu_F(u,x)$	(atanU u x	†
$arccotu_F(u,x)$	(acotU u x	†
$arcctgu_F(u,x)$	(* (sign x) (acotU u (abs x)))	†
$arcsecu_F(u,x)$	(asecU u x)	†
$arccscu_F(u,x)$	(acscU u x)	†
$arcu_F(u,x,y)$	(atan2U u y x)	†
$deg\_nearest\_axis_F(u,x)$	(cycle_nearest_axis 360 x)	†
$deg\_offset\_axis_F(u,x)$	(cycle_offset_axis 360 x)	†
$deg_F(x)$	(cycle 360 x)	†

$sind_F(x)$	`(sinU 360` x`)`	†
$cosd_F(x)$	`(cosU 360` x`)`	†
$tand_F(x)$	`(tanU 360` x`)`	†
$cotd_F(x)$	`(cotU 360` x`)`	†
$secd_F(x)$	`(secU 360` x`)`	†
$cscd_F(x)$	`(cscU 360` x`)`	†
$arcsind_F(x)$	`(asinU 360` x`)`	†
$arccosd_F(x)$	`(acosU 360` x`)`	†
$arctand_F(x)$	`(atanU 360` x`)`	†
$arccotd_F(x)$	`(acotU 360` x`)`	†
$arcctgd_F(x)$	`(* (sign` x`) (acotU 360 (abs` x`)))`	†
$arcsecd_F(x)$	`(asecU 360` x`)`	†
$arccscd_F(x)$	`(acscU 360` x`)`	†
$arcd_F(x,y)$	`(atan2U 360` y x`)`	†
$rad\_to\_cycle_F(x,u)$	`(rad_to_cycle` x u`)`	†
$cycle\_to\_rad_F(u,x)$	`(cycle_to_rad` u x`)`	†
$cycle\_to\_cycle_F(u,x,v)$	`(cycle_to_cycle` u x v`)`	†

where b, x, y, u, and v are expressions of type FLT, and z is an expressions of type INT.

Type conversions in Common Lisp and in ISLisp...

$convert_{I\to I'}(x)$	x (only one integer type)	
$rounding_{F\to I}(y)$	`(round` y`)`	
$floor_{F\to I}(y)$	`(floor` y`)`	
$ceiling_{F\to I}(y)$	`(ceiling` y`)`	
$cvtnearest_{I\to F}(x)$		†
$cvtdown_{I\to F}(x)$		†
$cvtup_{I\to F}(x)$		†
$cvtnearest_{F\to F'}(y)$		†
$cvtdown_{F\to F'}(y)$		†
$cvtup_{F\to F'}(y)$		†
$cvtnearest_{F\to D}(y)$		†
$cvtdown_{F\to D}(y)$		†
$cvtup_{F\to D}(y)$		†
$cvtnearest_{D\to F}(z)$		†
$cvtdown_{D\to F}(z)$		†
$cvtup_{D\to F}(z)$		†

where x is an expressions of type INT, y is an expressions of type FLT, and z is an expressions of type FXD, where FXD is a fixed point type.

Common List and ISLisp provides non-negative base 10 numerals for all its integer and floating

point types. There is no differentiation between the numerals for different floating point datatypes, nor between numerals for different integer types, and integer numerals can be used for floating point values. The details are not repeated here, see ....

C.8 Modula 2

The programming language Modula-2 is defined by ISO/IEC , Information Technology – Programming Languages – Modula-2 .

An implementation should follow all the requirements of LIA-2 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the LIA-2 for that operation. For each of the marked items a suggested identifier is provided.

The Modula-2 datatype `Boolean` corresponds to the LIA-1 datatype *Boolean*.

The additional integer operations are listed below, along with the syntax used to invoke them:

$min_I(x,y)$	`Imin(`x`, `y`)`	†
$max_I(x,y)$	`Imax(`x`, `y`)`	†
$min\_seq_I(xs)$	`IminArr(`xs`)`	†
$max\_seq_I(xs)$	`ImaxArr(`xs`)`	†
$dim_I(x,y)$	`Idim(`x`, `y`)`	†
$sqrt_I(x)$	`Isqrt(`x`)`	†
$power_I(x,y)$	`Ipower(`x`, `y`)`	†
$divides_I(x,y)$	`Divides(`x`, `y`)`	†
$even_I(x)$	`(not Odd(`x`))`	
$odd_I(x)$	`Odd(`x`)`	
$gcd_I(x,y)$	`Gcd(`x`, `y`)`	†
$lcm_I(x,y)$	`Lcm(`x`, `y`)`	†
$gcd\_seq_I(xs)$	`GcdArr(`xs`)`	†
$lcm\_seq_I(xs)$	`LcmArr(`xs`)`	†
$add\_wrap_I(x,y)$	`AddWrap(`x`, `y`)`	†
$add\_ov_I(x,y)$	`AddOver(`x`, `y`)`	†
$sub\_wrap_I(x,y)$	`SubWrap(`x`, `y`)`	†
$sub\_ov_I(x,y)$	`SubOver(`x`, `y`)`	†
$mul\_wrap_I(x,y)$	`MulWrap(`x`, `y`)`	†
$mul\_ov_I(x,y)$	`MulOver(`x`, `y`)`	†

where x and y are expressions of type *INT* and where xs is an expression of type `array [] of` *INT*.

The additional non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$min_F(x,y)$	`Min(`x`, `y`)`	†
$max_F(x,y)$	`Max(`x`, `y`)`	†
$min\_seq_F(xs)$	`MinArr(`xs`)`	†
$max\_seq_F(xs)$	`MaxArr(`xs`)`	†
$rounding_F(x)$	`Rounding(`x`)`	†

$floor_F(x)$	`Floor(`x`)`	†
$ceiling_F(x)$	`Ceiling(`x`)`	†
$dim_F(x,y)$	`Dim(`x`, `y`)`	†
$add3_F(x,y,z)$	`Add(`x`, `y`, `z`)`	†
$sum_F(xs)$	`Sum(`xs`)`	†
$dprod_{F\to F'}(x,y)$	`Prod(`x`, `y`)`	†
$mul\_add_F(x,y,z)$	`MulAdd(`x`, `y`, `z`)`	†
$irem_F(x,y)$	`Remainder(`x`, `y`)`	†
$sqrt_F(x)$	`Sqrt(`x`)`	
$rsqrt_F(x)$	`Rsqrt(`x`)`	†
$add\_lo_F(x,y)$	`AddLow(`x`, `y`)`	†
$sub\_lo_F(x,y)$	`SubLow(`x`, `y`)`	†
$mul\_lo_F(x,y)$	`MulLow(`x`, `y`)`	†
$div\_rest_F(x,y)$	`DivRest(`x`, `y`)`	†
$sqrt\_rest_F(x)$	`SqrtRest(`x`)`	†

where x, y and z are expressions of type *FLT*, and where xs is an expression of type `array [] of` *FLT*.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$max\_err\_hypot_F$	`Err_hypotenuse(`x`)`	†
$max\_err\_exp_F$	`Err_exp(`x`)`	†
$max\_err\_power_F(b,x)$	`Err_power(`b`, `x`)`	†
$max\_err\_sinh_F$	`Err_sinh(`x`)`	†
$max\_err\_tanh_F$	`Err_tanh(`x`)`	†
$big\_radian\_angle_F$	`Big_radian_angle(`x`)`	†
$max\_err\_sin_F$	`Err_sin(`x`)`	†
$max\_err\_tan_F$	`Err_tan(`x`)`	†
big_angle_F	`Big_angle(`x`)`	†
$max\_err\_sinu_F(u)$	`Err_sin_cycle(`u`)`	†
$max\_err\_tanu_F(u)$	`Err_tan_cycle(`u`)`	†

where b, x and u are expressions of type *FLT*. Several of the parameter functions are constant for each type (and library), the argument is then used only to differentiate among the floating point types.

$hypot_F(x,y)$	`Hypotenuse(`x`, `y`)`	†
$exp_F(x)$	`Exp(`x`)`	
$expm1_F(x)$	`ExpM1(`x`)`	†
$power_{FI}(b,z)$	`PowerI(`b`, `z`)`	†
$power_F(b,y)$	`Power(`b`, `y`)`	†
$powerm1_F(b,y)$	`PowerM1(`b`, `y`)`	†
$exp2_F(x)$	`Exp2(`x`)`	†
$exp10_F(x)$	`Exp10(`x`)`	†

$ln_F(x)$	`Ln(`x`)`	
$ln1p_F(x)$	`Ln1P(`x`)`	†
$log_F(b,x)$	`Log(`x`, `b`)`	†
$log1p_F(b,x)$	`Log1P(`x`, `b`)`	†
$log2_F(x)$	`Log2(`x`)`	†
$log10_F(x)$	`Log10(`x`)`	†
$sinh_F(x)$	`Sinh(`x`)`	†
$cosh_F(x)$	`Cosh(`x`)`	†
$tanh_F(x)$	`Tanh(`x`)`	†
$coth_F(x)$	`Coth(`x`)`	†
$sech_F(x)$	`Sech(`x`)`	†
$csch_F(x)$	`Csch(`x`)`	†
$arcsinh_F(x)$	`Arcsinh(`x`)`	†
$arccosh_F(x)$	`Arccosh(`x`)`	†
$arctanh_F(x)$	`Arctanh(`x`)`	†
$arccoth_F(x)$	`Arccoth(`x`)`	†
$arcsech_F(x)$	`Arcsech(`x`)`	†
$arccsch_F(x)$	`Arccsch(`x`)`	†
$rad\_nearest\_axis_F(x)$	`nearest_axis(`x`)`	†
$rad\_offset\_axis_F(x)$	`offset_axis(`x`)`	†
$rad_F(x)$	`Radian(`x`)`	†
$sin_F(x)$	`Sin(`x`)`	
$cos_F(x)$	`Cos(`x`)`	
$tan_F(x)$	`Tan(`x`)`	†
$cot_F(x)$	`Cot(`x`)`	†
$sec_F(x)$	`Sec(`x`)`	†
$csc_F(x)$	`Csc(`x`)`	†
$arcsin_F(x)$	`Arcsin(`x`)`	†
$arccos_F(x)$	`Arccos(`x`)`	†
$arctan_F(x)$	`Arctan(`x`)`	
$arccot_F(x)$	`Arccot(`x`)`	†
$arcctg_F(x)$	`(Sign(`x`)*Arccot(Abs(`x`)))`	†
$arcsec_F(x)$	`Arcsec(`x`)`	†
$arccsc_F(x)$	`Arccsc(`x`)`	†
$arc_F(x,y)$	`Angle(`x`, `y`)`	†
$cycle\_nearest\_axis_F(u,x)$	`nearest_axisU(`u`, `x`)`	†
$cycle\_offset\_axis_F(u,x)$	`offset_axisU(`u`, `x`)`	†
$cycle_F(u,x)$	`Cycle(`u`, `x`)`	†
$sinu_F(u,x)$	`SinU(`u`, `x`)`	†
$cosu_F(u,x)$	`CosU(`u`, `x`)`	†
$tanu_F(u,x)$	`TanU(`u`, `x`)`	†
$cotu_F(u,x)$	`CotU(`u`, `x`)`	†

$secu_F(u,x)$	SecU(u, x)	†
$cscu_F(u,x)$	CscU(u, x)	†
$arcsinu_F(u,x)$	ArcsinU(u, x)	†
$arccosu_F(u,x)$	ArccosU(u, x)	†
$arctanu_F(u,x)$	ArctanU(u, x)	†
$arccotu_F(u,x)$	ArccotU(u, x)	†
$arcctgu_F(u,x)$	(Sign(x)*ArccotU(u, Abs(x)))	†
$arcsecu_F(u,x)$	ArcsecU(u, x)	†
$arccscu_F(u,x)$	ArccscU(u, x)	†
$arcu_F(u,x,y)$	AngleU(u, x, y)	†
$sind_F(x)$	SinU(360.0, x)	†
$cosd_F(x)$	CosU(360.0, x)	†
$tand_F(x)$	TanU(360.0, x)	†
$cotd_F(x)$	CotU(360.0, x)	†
$secd_F(x)$	SecU(360.0, x)	†
$cscd_F(x)$	CscU(360.0, x)	†
$deg\_nearest\_axis_F(u,x)$	nearest_axisU(360.0, x)	†
$deg\_offset\_axis_F(u,x)$	offset_axisU(360.0, x)	†
$deg_F(x)$	Cycle(360.0, x)	†
$arcsind_F(x)$	ArcsinU(360.0, x)	†
$arccosd_F(x)$	ArccosU(360.0, x)	†
$arctand_F(x)$	ArctanU(360.0, x)	†
$arccotd_F(x)$	ArccotU(360.0, x)	†
$arcctgd_F(x)$	(Sign(x)*ArccotU(360.0, Abs(x)))	†
$arcsecd_F(x)$	ArcsecU(360.0, x)	†
$arccscd_F(x)$	ArccscU(360.0, x)	†
$arcd_F(x,y)$	AngleU(360.0, x,y)	†
$rad\_to\_cycle_F(x,u)$	Radian_to_cycle(x, u)	†
$cycle\_to\_rad_F(u,x)$	Cycle_to_radian(u, x)	†
$cycle\_to\_cycle_F(u,x,v)$	Cycle_to_cycle(u, x, v)	†

where b, x, y, u, and v are expressions of type FLT, and z is an expressions of type INT

....

$convert_{I\to I'}(x)$		†
$rounding_{F\to I}(y)$	Round(y)	†
$floor_{F\to I}(y)$		
$ceiling_{F\to I}(y)$		
$cvtnearest_{I\to F}(x)$		†
$cvtdown_{I\to F}(x)$		†
$cvtup_{I\to F}(x)$		†

$cvtnearest_{F \to F'}(y)$	†
$cvtdown_{F \to F'}(y)$	†
$cvtup_{F \to F'}(y)$	†
$cvtnearest_{F \to D}(y)$	†
$cvtdown_{F \to D}(y)$	†
$cvtup_{F \to D}(y)$	†
$cvtnearest_{D \to F}(z)$	†
$cvtdown_{D \to F}(z)$	†
$cvtup_{D \to F}(z)$	†

where x is an expressions of type *INT*, y is an expressions of type *FLT*, and z is an expressions of type *FXD*, where *FXD* is a fixed point type.

Modula-2 provides non-negative numerals .....

C.9 Pascal and Extended Pascal

The programming language Extended Pascal is defined in ISO/IEC 10206:1991 *Information technology – Programming languages – Extended Pascal* [11]. The programming language ISO Pascal is defined by ISO/IEC 7185:1990, *Information technology – Programming languages – Pascal* [5]. The programming language ANSI/IEEE Pascal is defined in ANSI/IEEE 770/X3.97-1983 [17].

An implementation should follow all the requirements of LIA-2 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the LIA-2 for that operation. For each of the marked items a suggested identifier is provided.

The Pascal datatype `Boolean` corresponds to the LIA-1 datatype *Boolean*.

The additional integer operations are listed below, along with the syntax used to invoke them:

$min_I(x, y)$	`Imin(`x`, `y`)`	†
$max_I(x, y)$	`Imax(`x`, `y`)`	†
$min\_seq_I(xs)$	`IminArr(`xs`)`	†
$max\_seq_I(xs)$	`ImaxArr(`xs`)`	†
$dim_I(x, y)$	`Idim(`x`, `y`)`	†
$sqrt_I(x)$	`Isqrt(`x`)`	†
$power_I(x, y)$	`Ipower(`x`, `y`)`	†
$divides_I(x, y)$	`Divides(`x`, `y`)`	†
$even_I(x)$	`(not Odd(`x`))`	
$odd_I(x)$	`Odd(`x`)`	
$gcd_I(x, y)$	`Gcd(`x`, `y`)`	†
$lcm_I(x, y)$	`Lcm(`x`, `y`)`	†
$gcd\_seq_I(xs)$	`GcdArr(`xs`)`	†
$lcm\_seq_I(xs)$	`LcmArr(`xs`)`	†
$add\_wrap_I(x, y)$	`AddWrap(`x`, `y`)`	†
$add\_ov_I(x, y)$	`AddOver(`x`, `y`)`	†

$sub\_wrap_I(x, y)$	`SubWrap(`x`, `y`)`	†
$sub\_ov_I(x, y)$	`SubOver(`x`, `y`)`	†
$mul\_wrap_I(x, y)$	`MulWrap(`x`, `y`)`	†
$mul\_ov_I(x, y)$	`MulOver(`x`, `y`)`	†

where x and y are expressions of type *INT* and where xs is an expression of type `array of` *INT*.

The additional non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$min_F(x, y)$	`Min(`x`, `y`)`	†
$max_F(x, y)$	`Max(`x`, `y`)`	†
$min\_seq_F(xs)$	`MinArr(`xs`)`	†
$max\_seq_F(xs)$	`MaxArr(`xs`)`	†
$rounding_F(x)$	`Rounding(`x`)`	†
$floor_F(x)$	`Floor(`x`)`	†
$ceiling_F(x)$	`Ceiling(`x`)`	†
$dim_F(x, y)$	`Dim(`x`, `y`)`	†
$add3_F(x, y, z)$	`Add(`x`, `y`, `z`)`	†
$sum_F(xs)$	`Sum(`xs`)`	†
$dprod_{F \to F'}(x, y)$	`Prod(`x`, `y`)`	†
$mul\_add_F(x, y, z)$	`MulAdd(`x`, `y`, `z`)`	†
$irem_F(x, y)$	`Remainder(`x`, `y`)`	†
$sqrt_F(x)$	`Sqrt(`x`)`	
$rsqrt_F(x)$	`Rsqrt(`x`)`	†
$add\_lo_F(x, y)$	`AddLow(`x`, `y`)`	†
$sub\_lo_F(x, y)$	`SubLow(`x`, `y`)`	†
$mul\_lo_F(x, y)$	`MulLow(`x`, `y`)`	†
$div\_rest_F(x, y)$	`DivRest(`x`, `y`)`	†
$sqrt\_rest_F(x)$	`SqrtRest(`x`)`	†

where x, y and z are expressions of type *FLT*, and where xs is an expression of type `array of` *FLT*.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$max\_err\_hypot_F$	`Err_hypotenuse(`x`)`	†
$max\_err\_exp_F$	`Err_exp(`x`)`	†
$max\_err\_power_F(b, x)$	`Err_power(`b`, `x`)`	†
$max\_err\_sinh_F$	`Err_sinh(`x`)`	†
$max\_err\_tanh_F$	`Err_tanh(`x`)`	†
$big\_radian\_angle_F$	`Big_radian_angle(`x`)`	†
$max\_err\_sin_F$	`Err_sin(`x`)`	†
$max\_err\_tan_F$	`Err_tan(`x`)`	†
big_angle_F	`Big_angle(`x`)`	†
$max\_err\_sinu_F(u)$	`Err_sin_cycle(`u`)`	†

$max\_err\_tanu_F(u)$	`Err_tan_cycle(`u`)`	†

where b, x and u are expressions of type *FLT*. Several of the parameter functions are constant for each type (and library), the argument is then used only to differentiate among the floating point types.

$hypot_F(x, y)$	`Hypotenuse(`x`, `y`)`	†
$exp_F(x)$	`Exp(`x`)`	
$expm1_F(x)$	`ExpM1(`x`)`	†
$power_{FI}(b, z)$	`PowerI(`b`, `z`)`	†
$power_F(b, y)$	`Power(`b`, `y`)`	†
$powerm1_F(b, y)$	`PowerM1(`b`, `y`)`	†
$exp2_F(x)$	`Exp2(`x`)`	†
$exp10_F(x)$	`Exp10(`x`)`	†
$ln_F(x)$	`Ln(`x`)`	
$ln1p_F(x)$	`Ln1P(`x`)`	†
$log_F(b, x)$	`Log(`x`, `b`)`	†
$log1p_F(b, x)$	`Log1P(`x`, `b`)`	†
$log2_F(x)$	`Log2(`x`)`	†
$log10_F(x)$	`Log10(`x`)`	†
$sinh_F(x)$	`Sinh(`x`)`	†
$cosh_F(x)$	`Cosh(`x`)`	†
$tanh_F(x)$	`Tanh(`x`)`	†
$coth_F(x)$	`Coth(`x`)`	†
$sech_F(x)$	`Sech(`x`)`	†
$csch_F(x)$	`Csch(`x`)`	†
$arcsinh_F(x)$	`Arcsinh(`x`)`	†
$arccosh_F(x)$	`Arccosh(`x`)`	†
$arctanh_F(x)$	`Arctanh(`x`)`	†
$arccoth_F(x)$	`Arccoth(`x`)`	†
$arcsech_F(x)$	`Arcsech(`x`)`	†
$arccsch_F(x)$	`Arccsch(`x`)`	†
$sin_F(x)$	`Sin(`x`)`	
$cos_F(x)$	`Cos(`x`)`	
$tan_F(x)$	`Tan(`x`)`	†
$cot_F(x)$	`Cot(`x`)`	†
$sec_F(x)$	`Sec(`x`)`	†
$csc_F(x)$	`Csc(`x`)`	†
$rad\_nearest\_axis_F(x)$	`nearest_axis(`x`)`	†
$rad\_offset\_axis_F(x)$	`offset_axis(`x`)`	†
$rad_F(x)$	`Radian(`x`)`	†
$arcsin_F(x)$	`Arcsin(`x`)`	†
$arccos_F(x)$	`Arccos(`x`)`	†
$arctan_F(x)$	`Arctan(`x`)`	

$arccot_F(x)$	Arccot(x)	†
$arcctg_F(x)$	(Sign(x)*Arccot(Abs(x)))	†
$arcsec_F(x)$	Arcsec(x)	†
$arccsc_F(x)$	Arccsc(x)	†
$arc_F(x,y)$	Angle(x, y)	†
$sinu_F(u,x)$	SinU(u, x)	†
$cosu_F(u,x)$	CosU(u, x)	†
$tanu_F(u,x)$	TanU(u, x)	†
$cotu_F(u,x)$	CotU(u, x)	†
$secu_F(u,x)$	SecU(u, x)	†
$cscu_F(u,x)$	CscU(u, x)	†
$cycle\_nearest\_axis_F(u,x)$	nearest_axisU(u, x)	†
$cycle\_offset\_axis_F(u,x)$	offset_axisU(u, x)	†
$cycle_F(u,x)$	Cycle(u, x)	†
$arcsinu_F(u,x)$	ArcsinU(u, x)	†
$arccosu_F(u,x)$	ArccosU(u, x)	†
$arctanu_F(u,x)$	ArctanU(u, x)	†
$arccotu_F(u,x)$	ArccotU(u, x)	†
$arcctgu_F(u,x)$	(Sign(x)*ArccotU(u, Abs(x)))	†
$arcsecu_F(u,x)$	ArcsecU(u, x)	†
$arccscu_F(u,x)$	ArccscU(u, x)	†
$arcu_F(u,x,y)$	AngleU(u, x, y)	†
$sind_F(x)$	SinU(360.0, x)	†
$cosd_F(x)$	CosU(360.0, x)	†
$tand_F(x)$	TanU(360.0, x)	†
$cotd_F(x)$	CotU(360.0, x)	†
$secd_F(x)$	SecU(360.0, x)	†
$cscd_F(x)$	CscU(360.0, x)	†
$deg\_nearest\_axis_F(u,x)$	nearest_axisU(360.0, x)	†
$deg\_offset\_axis_F(u,x)$	offset_axisU(360.0, x)	†
$deg_F(x)$	Cycle(360.0, x)	†
$arcsind_F(x)$	Arcsin(360.0, x)	†
$arccosd_F(x)$	Arccos(360.0, x)	†
$arctand_F(x)$	Arctan(360.0, x)	†
$arccotd_F(x)$	Arccot(360.0, x)	†
$arcctgd_F(x)$	(Sign(x)*Arccot(360.0, Abs(x)))	†
$arcsecd_F(x)$	Arcsec(360.0, x)	†
$arccscd_F(x)$	Arccsc(360.0, x)	†
$arcd_F(x,y)$	Angle(360.0, x,y)	†
$rad\_to\_cycle_F(x,u)$	RadianToCycle(x, u)	†
$cycle\_to\_rad_F(u,x)$	CycleToRadian(u, x)	†
$cycle\_to\_cycle_F(u,x,v)$	CycleToCycle(u, x, v)	†

where b, x, y, u, and v are expressions of type *FLT*, and z is an expressions of type *INT*

....

$convert_{I \to I'}(x)$		†
$floor_{F \to I}(y)$	`Floor`(y)	
$rounding_{F \to I}(y)$	`Round`(y)	†
$ceiling_{F \to I}(y)$		
$cvtdown_{I \to F}(x)$		†
$cvtnearest_{I \to F}(x)$		†
$cvtup_{I \to F}(x)$		†
$cvtdown_{F \to F'}(y)$		†
$cvtnearest_{F \to F'}(y)$		†
$cvtup_{F \to F'}(y)$		†
$cvtdown_{F \to D}(y)$		†
$cvtnearest_{F \to D}(y)$		†
$cvtup_{F \to D}(y)$		†
$cvtdown_{D \to F}(z)$		†
$cvtnearest_{D \to F}(z)$		†
$cvtup_{D \to F}(z)$		†

where x is an expressions of type *INT*, y is an expressions of type *FLT*, and z is an expressions of type *FXD*, where *FXD* is a fixed point type.

Pascal provides non-negative numerals .....

Annex D
(informative)

Bibliography

This annex gives references to publications relevant to ISO/IEC 109672.

International Standards Documents

[1] IEC 559:1989, *Binary floating-point arithmetic for microprocessor systems.* (Also: ANSI/IEEE Std 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic.*)

[2] ISO/IEC Directives, Part 3: *Drafting and presentation of International Standards*, 1989.

[3] ISO/IEC 1539:1991, *Information technology – Programming languages – Fortran.* (Also: ANSI X3.198-1991, *American National Standard Programming Language Fortran.*)

[4] ISO/IEC 6160:1979, *Information technology – Programming languages – PL/I.* (Endorsement of ANSI X3.53-1976, *American National Standard Programming Language PL/I.*)

[5] ISO/IEC 7185:1990, *Information technology – Programming languages – Pascal.* (Also: ANSI/IEEE 770/X3.97-1983, *American National Standard / IEEE Standard / Pascal Computer Programming Language.*)

[6] ISO/IEC 8652:1995, *Information technology – Programming languages – Ada.*

[7] ISO/IEC 8825:1990, *Information Processing Systems – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One* (ASN.1).

[8] ISO/IEC 9001:1987, *Quality systems – Model for quality assurance in production and installation.*

[9] ISO/IEC 9899:1990, *Information technology – Programming languages – C.* (Also: ANSI X3.159-1989, *American National Standard for Information Systems – Programming Language – C.*)

[10] ISO/IEC TR 10176:1991, *Information technology – Guidelines for the preparation of programming language standards.*

[11] ISO/IEC 10206:1991, *Information technology – Programming languages – Extended Pascal.* (Also: ANSI/IEEE 770/X3.160-1989, *Standard for the programming language Extended Pascal.*)

[12] ISO/IEC 10279:1991, *Information technology – Programming languages for Information Systems – Programming Languages – Full BASIC.* (Also: ANSI X3.113-1987 *American National Standard for Information Systems – Programming Languages – Full BASIC.*)

[13] ISO/IEC 10514:1994, *Information technology – Programming languages – Modula-2.*

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

[15] ISO/IEC 10967-3:—, *Information technology – Language independent arithmetic – Part 3: Complex floating point arithmetic and complex elementary numerical functions.* (To be published.)

National Standards Documents

[16] ANSI/MIL-STD-1815A-1995?, *Reference Manual for the Ada Programming Language.*

[17] ANSI/IEEE 770/X3.97-1983, *American National Standard / IEEE Standard / Pascal Computer Programming Language.*

[18] ANSI/IEEE 770/X3.160-1989, *Standard for the programming language Extended Pascal.*

[19] ANSI X3.113-1987 *American National Standard for Information Systems – Programming Languages – Full BASIC.*

[20] ANSI X3.159-1989, *American National Standard for Information Systems – Programming Language – C.*

[21] ANSI X3.198-1991, *American National Standard Programming Language Fortran.*

[22] ANSI/IEEE Std 754-1984, *IEEE Standard for Binary Floating-Point Arithmetic.*

[23] ANSI/IEEE Std 854-1987, *A Radix-Independent Standard for Floating-Point Arithmetic.*

[24] ANSI X3.226, *American National Standard for Information Systems – Programming Language – Common Lisp* (Draft 12.24).

[25] ANSI X3.74-1987, *Americal National Standard Programming Language General Purpose PL/I.*

Books, Articles, and Other Documents

[26] J S Squire (ed), *Ada Letters*, vol. XI, No. 7, ACM Press (1991).

[27] M Abramowitz and I Stegun (eds), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, Tenth Printing, 1972, Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

[28] J Du Croz and M Pont, *The Development of a Floating-Point Validation Package*, *NAG Newsletter*, No. 3, 1984.

[29] J W Demmel and X Li, *Faster Numerical Algorithms via Exception Handling*, 11th International Symposium on Computer Arithmetic, Winsor, Ontario, June 29 - July 2, 1993.

[30] D Goldberg, *What Every Computer Scientist Should Know about Floating-Point Arithmetic.* ACM Computing Surveys, Vol. 23, No. 1, March 1991.

[31] J R Hauser, *Handling Floating-Point Exceptions in Numeric Programs.* ACM Transactions on Programming Languages and Systems, Vol. 18, No. 2, March 1986, Pages 139-174.

[32] W Kahan, *Branch Cuts for Complex Elementary Functions, or Much Ado about Nothing's Sign Bit*, Chapter 7 in *The State of the Art in Numerical Analysis* ed. by M Powell and A Iserles (1987) Oxford.

[33] W Kahan, *Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic*, Panel Discussion of *Floating-Point Past, Present and Future*, May 23, 1995, in a series of San Francisco Bay Area Computer Historical Perspectives, sponsored by SUN Microsystems Inc.

[34] U Kulisch and W L Miranker, *Computer Arithmetic in Theory and Practice*, Academic Press, 1981.

[35] U Kulisch and W L Miranker (eds), *A New Approach to Scientific Computation*, Academic Press, 1983.

[36] D C Sorenson and P T P Tang, *On the Orthogonality of Eigenvectors Computed by Divide-and-Conquer Techniques*, SIAM Journal of Numerical Analysis, Vol. 28, No. 6, p. 1760, algorithm 5.3.

[37] *Floating-Point C Extensions* in Technical Report Numerical C Extensions Committee X3J11, April 1995, SC22/WG14 N403, X3J11/95-004.

[38] M Payne and R Hanek, *Radian Reduction for Trigonometric Functions*, SIGNUM Newsletter, Vol. 18, January 1983.

[39] M Payne and R Hanek, *Degree Reduction for Trigonometric Functions*, SIGNUM Newsletter, Vol. 18, April 1983.

[40] N L Schryer, *A Test of a Computer's Floating-Point Unit*, Computer Science Technical Report No. 89, AT&T Bell Laboratories, Murray Hill, NJ, 1981.