J3/97-199

Date: July 23, 1997
From: Dmitri Chiriaev and G. William Walster
To: J3
Subject: Interval Arithmetic Specifications

Abstract

To provide compiler support for interval arithmetic requires a specification of both the required syntax and semantics. The specification contained herin builds on Priest's [10] definition of valid floating–point intervals and arithmetic operations that yield a closed and consistent system even in the presence of exceptions. In this paper, Prist's ideas are extended to include the correct nonstop handling of empty intervals and IEEE 754 exceptions in all interval relational operators and intrinsics. The process of considering what is possible at compile and run-time has led to some new results in interval mathematics [12]. These new results motivate introduction of some new operators and interval intrinsics.

Contents

1 Introduction

Priest [10] defines a set of representable real intervals based on the IEEE 754 floating–point number system together with efficient algorithms for computing the basic arithmetic operations and the square root function. These algorithms yield a consistent closed set of interval results even in the presence of exceptions. In this paper algorithms are defined that provide correct nonstop handling of IEEE 754 exceptions for all the interval intrinsics and operations defined in [4] as well as a number of new operators and intrinsics defined in [12] which are marked with ^{New}. All intrinsics defined in this paper are pure.

2 Interval type

In general a real interval, or just an interval, $[\underline{x}, \overline{x}]$ is a closed, bounded subset of the real numbers $\mathbb{R}$ of the form

$$X \equiv [\underline{x}, \overline{x}] \equiv \{x \in \mathbb{R} | \underline{x} \leq x \leq \overline{x}\} \quad ,$$

where $\underline{x}$ and $\overline{x}$ denote the *lower* and the *upper bounds* of the interval X, respectively. The set of real intervals is denoted by $I\mathbb{R}$. An interval is called a *point or degenerate* interval if $\underline{x} = \overline{x} = x$.

Intervals are opaque. Therefore users cannot directly access memory to set or retrieve the value of an interval. The endpoints of an interval type are each of the same real type available on the processor. The individual endpoints are accessible through the functions INF (9.2) and SUP (9.3) defined below. The entire interval can be set through the INTERVAL, CONVERT_DECIMAL_DIGITS or CONVERT_WITHIN_BOUNDS (see 18).

The precision of each interval data type shall correspond to the precision of a real data type. The kind type parameter of each supported interval type shall be the same as the kind type parameter of the corresponding real type.

3 Interval constants

Where literal constants are admitted in a program interval's may be represented by one of the following:

- a single real or integer

- a single real or a pair of real's, integer's, beginning with "[", separated by "," if there are two numbers, and ending in "]".

As input or output data, interval's may be represented by one of the following:

- a single real or integer
- a pair of real's, integer's,
- combinations thereof, beginning with "[", separated by "," if there are two numbers, and ending in "]". For single number (see sec. 19.3) input [and] are optional.

For example

[1,2], [1E0,3], [1], and [.1234D5]

are all valid interval constants. Though the use of "[" and "]" to denote the end-points of an interval constant (in contrast to "(<" and ">)" proposed in [4]) introduces an additional character to the Fortran character set, it simplifies the parsing of a program containing interval constants and coincides with the usual mathematical notation for intervals. It is clearly nicer aesthetically.

An interval constant specified by a single number is the same as an interval constant specified by two numbers, both of whose endpoints are equal to the single number. When such a decimal constant is converted to its internal representation, the internal representation shall contain the decimal constant, regardless of how many digits are specified by the decimal constant.

4 Valid intervals

To get the set of extended real intervals $I\!I\!R^*$, $I\!I\!R$ is extended by adding the empty interval (4.1), and the intervals containing ideal points minus and plus infinity.

$$I\!I\!R^* \equiv I\!I\!R \cup \{[-\infty, z] : z \in I\!R\} \cup \{[z, +\infty] : z \in I\!R\} \cup \{[-\infty, +\infty]\} \cup \emptyset$$

Improper intervals with upper bound less than lower bound are invalid. Intervals $[-\infty, -\infty]$ and $[+\infty, +\infty]$, are not allowed to avoid the $\infty - \infty$ invalid operation exception in interval addition.

Priest [10] proposes to use the interval [NaN, NaN] to represent the empty interval and defines interval arithmetic algorithms in such a way that the result of any interval operation on the empty interval operand yields the empty interval result.

Definition 4.1 (Empty interval) *The interval* $X \equiv [\underline{x}, \overline{x}]$ *is empty interval* $\emptyset$*, if both* $\underline{x}$ *and* $\overline{x}$ *are a NaN.*

Some required properties of empty intervals:

- The empty interval is is a *subset* (10.3) of every other interval.
- The empty interval is *set equal* (11.1) to itself.
- The empty interval is *disjoint* (10.6) with any interval, including itself.
- The *intersection* (9.4) of the empty interval with any interval, including itself is empty.
- Any interval extension of a point function of one or more arguments is empty when any of its interval arguments is empty.

In IEEE 754 binary floating-point arithmetic [2] the result of any relational operation $<, \leq, >, \geq, =$ with a NaN as an argument is false. NaN is not equal to any floating–point number including itself.

Therefore for some operators special algorithms are necessary to provide the behavior expected for empty sets. The efficiency of these algorithms and algorithms for arithmetic operators can be increased through the efficient implementation of the min and max functions for the following special operands:

(i) $\min(x, y) = \min(y, x)$ for all x, y, and similarly for max,

(ii) $\min(x, \mathrm{NaN}) = \max(x, \mathrm{NaN}) = \mathrm{NaN}$ for any x, and

(iii) $\min(-0, 0) = -0$, $\max(-0, 0) = 0$.

See [10] for complete discussion.

5 Handling interval exceptions

The issue of properly handling IEEE 754 [2] exceptions in interval arithmetic was first addressed by Popova in [9]. The proposal for interval arithmetic in Fortran 90 [4] defines new interval relational operators and requires that all Fortran intrinsic functions that accept real data shall also accept interval data. The J3/97-155 [5] proposal discusses the exceptions for interval intrinsic functions.

The present specification is consistent with the proposal for floating–point exception handling as defined in [3] and extend the [5] by adding the INTERVAL_DIVIDE_BY_ZERO exception.

There shall be a data type INTERVAL_FLAG_TYPE, for identifying a particular exception flag. Its only possible values are those of named constants INTERVAL_OUT_OF_RANGE, INTERVAL_POSSIBLY_OUT_OF_RANGE and INTERVAL_DIVIDE_BY_ZERO.

5.1 The exceptions

INTERVAL_DIVIDE_BY_ZERO
This exception occurs when an interval division has a degenarate zero interval denominator.

INTERVAL_OUT_OF_RANGE
An argument of an interval intrinsic is totally outside the domain of definition.

INTERVAL_POSSIBLY_OUT_OF_RANGE
An argument of an interval intrinsic is partially outside the domain of definition.

Each exception has a flag whose value is either quiet or signalling. The value may be determined by the function INTERVAL_GET_FLAG. Its initial value is quiet and it signals when the associated exception occurs. Its status can be also changed by the subroutine INTERVAL_SET_FLAG. Once signalling, it remains signalling unless set quiet by an invocation of the subroutine INTERVAL_SET_FLAG. The value of the INTERVAL_POSSIBLY_OUT_OF_RANGE flag shall be true whenever the value of the INTERVAL_OUT_OF_RANGE flag is true.

5.2 Halting

Halting's control is exercised by invocation of the subroutine INTERVAL_SET_HALTING_MODE. Halting is not necessarily immediate and may occur any time after the exception has occured. In a procedure other than INTERVAL_SET_HALTING_MODE, the processor shall not change the halting mode on entry, and on return shall ensure that the halting mode is the same as it was on entry.

The initial value of the halting mode for all exception flags is false.

If the value of the halting mode INTERVAL_POSSIBLY_OUT_OF_RANGE is false, then the argument of the interval intrinsic function shall be intersected with the valid domain of the point intrinsic function. (The valid domain of an interval intrinsic function shall be the set of intervals contained in the valid domain of the corresponding floating–point intrinsic function.)

5.3 INTERVAL_GET_FLAG(FLAG, FLAG_VALUE)

Description. Get an exception flag

Class. Elemental subroutine.

Arguments. FLAG shall be of type TYPE(INTERVAL_FLAG_TYPE). It is an INTENT(IN) argument and specifies the INTERVAL flag to be obtained.

FLAG_VALUE shall be of default logical type. It is an INTENT(OUT) argument. If the value of FLAG is INTERVAL_DIVIDE_BY_ZERO, INTERVAL_OUT_OF_RANGE or INTERVAL_POSSIBLY_OUT_OF_RANGE, the result value is true if the corresponding exception flag is signaling and is false otherwise.

5.4 INTERVAL_SET_FLAG(FLAG,FLAG_VALUE)

Description. Assign a value to an exception flag

Class. Elemental subroutine.

Arguments. FLAG shall be of type TYPE(INTERVAL_FLAG_TYPE). It is an INTENT(IN) argument. If the value of FLAG is INTERVAL_DIVIDE_BY_ZERO, INTERVAL_OUT_OF_RANGE or INTERVAL_POSSIBLY_OUT_OF_RANGE, the corresponding exception flag is assigned a value.

FLAG_VALUE shall be of default logical type. It is an INTENT(IN) argument. If it has the value true, the flag is set to be signalling; otehrwise, the flag is set to be quiet.

5.5 INTERVAL_GET_HALTING_MODE(FLAG, HALTING)

Description. Get halting mode for an exception

Class. Elemental subroutine.

Arguments. FLAG shall be of type TYPE(INTERVAL_FLAG_TYPE). It is an INTENT(IN) argument and specifies the INTERVAL flag. It shall have one of the values INTERVAL_DIVIDE_BY_ZERO, INTERVAL_OUT_OF_RANGE or INTERVAL_POSSIBLY_OUT_OF_RANGE.

HALTING shall be scalar and of type default logical. It is an INTENT(OUT) argument. The value is true if the exception specified by FLAG will cause halting. Otherwise, the value is false.

5.6 INTERVAL_SET_HALTING_MODE(FLAG,HALTING)

Description. Controls continuation or halting after an interval exceptions

Class. Elemental subroutine.

Arguments. FLAG shall be of type TYPE(INTERVAL_FLAG_TYPE). It is an INTENT(IN) argument and specifies the INTERVAL flag. It shall have one of the values INTERVAL_DIVIDE_BY_ZERO, INTERVAL_OUT_OF_RANGE or INTERVAL_POSSIBLY_OUT_OF_RANGE.

HALTING shall be scalar and of type default logical. It is an INTENT(IN) argument. If it has the value true, the exception specified by flag will cause halting. Otherwise, execution will continue after this exception. The processor must either already be treating this exception in this way or be capable of changing the mode so that it does.

5.7 Examples

Given X=[0,1] and Y=[0], upon evaluation of X/Y the flag INTERVAL_DIVIDE_BY_ZERO is set to true.

Upon evaluation of SQRT([-1.0,-0.5]), both INTERVAL_OUT_OF_RANGE and INTERVAL_POSSIBLY_OUT_OF_RANGE are set to true.

Suppose INTERVAL_OUT_OF_RANGE and INTERVAL_POSSIBLY_OUT_OF_RANGE are both false immediately prior to evaluation of the expression SQRT ([-1.0,1.0]). Upon evaluation of SQRT([-1.0,1.0]), INTERVAL_OUT_OF_RANGE remains false but INTERVAL_POSSIBLY_OUT_OF_RANGE is set to true. If HALTING for INTERVAL_POSSIBLY_OUT_OF_RANGE is false, then execution continues, and the value of SQRT([-1.0,1.0]) is the same as the value of SQRT([-0.0,1.0]).

5.8 Handling the empty interval

The empty interval is a valid interval. Therefore no interval operation or function may signal the `Invalid Operation (IO)` exception, if any interval operand or argument is empty.

In the process of performing any interval operation with an empty argument any `IO` exception that may be generated by underlying floating-point operations must be cleared.

To extend the set of functions and operators for which exceptions can be handled in a non-stop way, the following rules are needed:

- A real-valued function of one or more interval argument will return a quiet NaN if any interval argument is empty. Exception: An empty argument of a function may be ignored if the function does not depend on the empty argument. For example $f(\emptyset) = c$ if $f(x) = c$. However $f(\emptyset) = \emptyset$, if $f(x) = x^0$, otherwise $f(x) = x^0 = 1$

- Any interval extension of a point function of one or more arguments will return the empty result when any of its interval arguments is empty, *provided the function depends on the empty argument.*

6 Mixed mode operations

Implicit conversion from interval to other data types, except to other kinds of interval, shall not be allowed.

The functions INF (9.2) and SUP (9.3) may be used to convert an interval type to another data type. Similarly, MID (14.2) may also be viewed as producing a real approximation to an interval, while INTERVAL (18.1) converts from real or integer to interval.

Mixed-mode INTERVAL/REAL and mixed mode INTERVAL/INTEGER operations shall be defined.

Note: Mixed-mode INTERVAL/COMPLEX is not defined.

Implicit conversion to interval shall be possible. The result of an implicit conversion to interval shall contain the mathematical interpretation of the original data type.

The implicit conversions shall be such that, at each conversion to interval, the resulting interval shall contain the exact mathematical value specified by the floating–point or integer variable. Conversion of literal constants to interval shall be as if the literal constant or constant expressions were converted directly to interval, with no intervening conversion to an internal floating–point representation.

Implicit conversion between intervals of one kind and another within expressions obeys the same rules as conversion between reals and integers and intervals. In expressions that contain more than one kind of interval, the conversion shall be to the kind corresponding to highest precision.

This strategy applied to expressions corresponds to the *widest need evaluation* strategy (see eg. [1]), according to which each arithmetic expression is evaluated using the widest format needed to evaluate any part of that expression.

7 Basic interval operations

An interval variable has a dual nature as both single unknown *number* and a *set* of single numbers. Many interval algorithms make use of this duality between sets and single numbers and combine set theoretic operations such as set intersection with arithmetic operations.

Interval constants are different, see [12]. An interval constant is a set of numerical values. The concept of dependence does not exist for interval constants. Interval constants are completely independent.

A fundamental problem of interval arithmetic is to compute the sharpest possible enclosure of a function f defined by an arithmetic expression $f(x)$ including arithmetic operations and intrinsic functions.

Substituting interval X for x in the definig expression of f one gets an *interval extension* of a function that usually overestimates the range. However there are cases when one really wishes to work with intervals as constants and the defining interval extension delivers the desired result.

The basic operations $\{+, -, *, /$ and $**\}$ are defined to contain the **ranges** of the corresponding operations on real numbers.

$$X \circ \ Y \supseteq \{x \circ y | x \in X, \ y \in Y\} \ \ \circ \in \{+, -, *\} \tag{1}$$

$$X/Y \supseteq \{x \circ y | x \in X, \ y \in Y \cap \{I\!R \setminus \{0\}\}\} \tag{2}$$

This definition of interval operations is equally suitable for dealing with intervals both as varibles and constants.

Arithmetic on constant intervals must be implemented using the above definitions to guarantee containment.

Arithmetic on dependent interval variables has been defined in [12].

$$X + X \supseteq \{x + x | x \in X\} \equiv \ [2 * \underline{x}, 2 * \overline{x}] \tag{3}$$

$$X - X \supseteq \{x - x | x \in X\} \equiv \ [0] \tag{4}$$

$$X * X \supseteq \{x * x | x \in X\} \equiv \begin{cases} [\min\{\underline{x}^2, \overline{x}^2\}, \max\{\underline{x}^2, \overline{x}^2\}] & : \text{ if } 0 < \underline{x} \text{ or } \overline{x} < 0 \\ [0, \max\{\underline{x}^2, \overline{x}^2\}] & : \text{ otherwise} \end{cases} \tag{5}$$

$$X/X \supseteq \{x/x | x \in X \cap \{\mathbb{R} \setminus \{0\}\}\} \equiv [1] \tag{6}$$

The overestimation incurred by the usual interval arithmetic definitions in (1) and (2) is eliminated by taking into account that the two operands are dependent. See [12] for justification of X/X being the degenerate interval [1], if $0 \in X$.

In the next version of this specification tables will be presented specifying the result of every interval arithmetic operation on every possible combination of *valid intervals* (see 4) .

The exponentiaton operator $X**Y$ is defined as

$$X**Y \supseteq \{x^y \mid x \in X \cap [0, +\infty], y \in Y\}.$$

For positive integer exponent $n \in \mathbb{N}$, $x**n$ is increasing for $x > 0$ and n even, or for all x with n odd, and decreasing otherwise. Thus, we get

$$X**n = \begin{cases} [\underline{x}^n, \overline{x}^n] & \text{if } 0 < \underline{x} \text{ or } n \text{ odd}, \\ [0, \max\{|\underline{x}|^n, |\overline{x}|^n] & \text{if } 0 \in X \text{ and } n \text{ even}, \\ [\overline{x}^n, \underline{x}^n] & \text{if } \overline{x} < 0 \text{ and } n \text{ even}. \end{cases}$$

For negative exponents with $0 \notin X$, $X^n = 1/X^{-n}$.
For a zero exponent, we set $X^0 = [1]$. Note: $\emptyset^0 = \emptyset$, but $0^0 = 1$.

As a special case we get an algorithm for $X**2$

$$X**2 = \begin{cases} [\min\{\underline{x}^2, \overline{x}^2\}, \max\{\underline{x}^2, \overline{x}^2\}] & \text{if } (0 < \underline{x} \text{ or } \overline{x} < 0) \\ [0, \max\{\underline{x}^2, \overline{x}^2\}] & \text{otherwise} \end{cases}$$

8 Optimization of interval expressions

The justification of interval expression folding can be found in [12]. The following is a brief overview of the discussion presented there.

The Fortran standard permits a compiler to substitute "mathematically equivalent" expressions within a given statement. However the "=" operator is defined to be an assignment of value, not a macro definition or an inlined function. This prohibits expression folding, which is nevertheless performed by optimizing compilers in the interest of run-time performance. To control expression folding there be two operators are required. They are $\equiv$ (inline function) and := (an assignment of value), respectively. Given current practice and the relative frequency with which $\equiv$ and := will be desired, it makes sense to permit the existing = operator be interpreted as $\equiv$

and introduce a new operator := to force assignment of value. With either a pragma or a command line option it must be possible to force both = and := to be interpreted as assignment of value to prohibit expression folding.

Interpreting PARAMETERs as read–only variables can lead to a containment failure (see [12]). For this reason, the default definition of the = operator in PARAMETER declarations in Fortran must be that of a macro definition. Only if the new assignment of value operator := is enabled, may PARAMETERs be treated as read–only variables.

9 Interval set intrinsics.

9.1 EMPTY(X)^{New}

Description. Tests if X is the empty interval.

$$EMPTY(X) \equiv \ (X = \emptyset)$$

Class. Inquiry function.

Argument. X shall be of type interval.

Result characteristics. Default logical scalar.

Result value. The result has the value true if X is empty (4.1) and otherwise has the value false.

Algorithm. not $(\underline{x} \leq \overline{x})$

Note.

To get the desired TRUE result if the operand is empty it is necessary to use the compliment of $(\underline{x} \leq \overline{x})$ rather than $(\underline{x} > \overline{x})$.

9.2 INF(X)

Description. Infimum of an interval.

Class. Elemental function.

Argument. X shall be of type interval.

Result characteristics. The result is of type real. The kind type parameter is that of X.

Result value. The result has the value $\underline{x}$ if X is not empty (4.1) and otherwise has the value of a quiet NaN.

Algorithm. if $\underline{x} \leq \overline{x}$ then $\underline{x}$ else NaN ;

9.3 SUP(X)

Description. Supremum of an interval.

Class. Elemental function.

Argument. X shall be of type interval.

Result characteristics. The result is of type real. The kind type parameter is that of X.

Result value. The result has the value $\overline{x}$ if X is not empty (4.1) and otherwise has the value of a quiet NaN.

Algorithm. if $\underline{x} \leq \overline{x}$ then $\overline{x}$ else NaN ;

9.4 Intersection (X.IS.Y)

Description. Computes the intersection of two intervals.

$$X \cap\ Y \equiv \{x : x \in X \text{ and } x \in Y\}$$

Class. Interval infix operation.

Arguments. X and Y shall be of type interval.

Result characteristics. The result is of type interval. The kind type parameter is determined by the types of the arguments according to rules for Fortran generic intrinsics (kind type parameter – corresponding to highest precision of arguments).

Result value. If either operand of the interval intersection operator is the empty interval then the result is the empty interval.

The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the description.

Algorithm.

$$\begin{array}{lcl} [\max\{\underline{x},\underline{y}\}, \min\{\overline{x},\overline{y}\}] & : & \text{if } \max\{\underline{x},\underline{y}\} \leq \min\{\overline{x},\overline{y}\} \\ \emptyset & : & \text{otherwise} \end{array}$$

9.5 Convex Hull (X .CH. Y)

Description. Computes the convex hull of two intervals

$$X.\mathrm{CH}.Y \equiv \begin{cases} \text{if } X = \emptyset & : \quad Y \\ \text{else if } Y = \emptyset & : \quad X \\ \text{else} & : \quad [\max\{inf(X), inf(Y)\}, \min\{sup(X), sup(Y)\}] \end{cases}$$

Class. Interval infix operation.

Argument. X and Y shall be of type interval.

Result characteristics. The result is of type interval. The kind type parameter is determined by the types of the arguments according to rules for Fortran generic intrinsics (kind type parameter – corresponding to highest precision of arguments).

Result value. If one of the operands of the interval convex hull operator is the empty interval then the result is the other operand.

The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the description.

Algorithm.

$$\begin{array}{lcl} \text{if } EMPTY(X) & : & [\underline{y}, \overline{y}] \\ \text{else if } EMPTY(Y) & : & [\underline{x}, \overline{x}] \\ \text{else} & : & [\max\{\underline{x}, \underline{y}\}, \min\{\overline{x}, \overline{y}\}] \end{array}$$

10 Set operations

10.1 Superset (X .SP. Y).

Description. Determines if X is a superset of Y.

$$X \supseteq Y \equiv (Y = \emptyset) \text{ or } (\forall y \in Y,\ \exists x' \in X,\ \exists x'' \in X :\ x' \leq y \leq x'')$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} \leq \underline{y}$ and $\overline{y} \leq \overline{x}$) or Y is the empty (4.1) interval, otherwise has the value false.

Algorithm. $\underline{x} \leq \underline{y}$ and $\overline{y} \leq \overline{x}$ or EMPTY(Y)

10.2 Proper superset (X.PSP.Y).

Description. Determines if X is a proper superset of Y

$$X \supset Y \equiv X \supseteq Y \text{ and } X.\text{SNE}.Y$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if X is a superset of Y and Y is unequal to X, otherwise the result has the value false.

Algorithm. ($\underline{x} \leq \underline{y}$ and $\overline{y} \leq \overline{x}$ and ($\underline{x} < \underline{y}$ or $\overline{y} < \overline{x}$) or (EMPTY(Y) and not EMPTY (X))

10.3 Subset (X.SB.Y).

Description. Determines if X is a subset of Y

$$X \subseteq Y \equiv (X = \emptyset) \text{ or } (\forall x \in X,\ \exists y' \in Y, \exists y'' \in Y :\ y' \leq x \leq y'')$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{y} \leq \underline{x}$ and $\overline{x} \leq \overline{y}$) or X is empty 4.1) otherwise has the value false.

Algorithm. $\underline{y} \leq \underline{x}$ and $\overline{x} \leq \overline{y}$ or EMPTY(X)

10.4 Proper subset (X.PSB.Y)

Description. Determines if X is a proper subset of Y

$$X \subset Y \equiv X \subseteq Y \text{ and } X.\text{SNE}.Y$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if X is a subset of Y and X is unequal to Y, otherwise the result has the value false.

Algorithm. ($\underline{x} \geq \underline{y}$ and $\overline{x} \leq \overline{y}$ and ($\underline{x} > \underline{y}$ or $\overline{x} < \overline{y}$)) or(EMPTY(X) and not EMPTY (Y))

10.5 In (X.IN.Y)

Description. Determines if the number is contained in the interval

$$x \in Y \equiv (\exists y' \in Y, \exists y'' \in Y : y' \leq x \leq y'')$$

Class. Relational interval operation.

Arguments.
X shall be of type integer or real.
Y shall be of type interval.

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{y} \leq x$ and $x \leq \overline{y}$) otherwise has the value false. The result has the value false if Y is empty (4.1)

Algorithm. $\underline{y} \leq x$ and $x \leq \overline{y}$

10.6 Disjoint (X.DJ.Y)

Description. Determines if two intervals are disjoint.

$$X.\text{DJ}.Y \equiv (\forall x \in X, \forall y \in Y : x \neq y)$$

The result shall be false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} > \overline{y}$ or $\overline{x} < \underline{y}$) or either X or Y or both is empty (4.1). Otherwise the result has the value false.

Algorithm. not ($\underline{x} \leq \overline{y}$ and $\underline{y} \leq \overline{x}$)

Required compile-time action. Not required, but possible.

> *From the definition it follows that X.DJ.X is FALSE if X is not empty. The proposed algorithm correctly enforces this result* **at run time time**, *but the check "EMPTY(X)" may be substituted for X.DJ.X .*

Note.

> *To get the desired true result if one or both arguments are empty it is necessary to use the compliment of ($\underline{x} \leq \overline{y}$ and $\underline{y} \leq \overline{x}$) rahter than ($\underline{x} > \overline{y}$ or $\overline{x} < \underline{y}$).*

11 Set relations

If the empty interval is an operand of a **set** relation then the result is false. The one exception is the set inequality (.SNE.) relation (11.2) which is true in this case.

11.1 Set equality. (X .SEQ. Y)

Description. Determines the equality of two intervals as sets.

$$X \text{ .SEQ.} Y \equiv (\forall x \in X,\ \exists y \in Y :\ x = y) \text{ and } (\forall y \in Y, \exists x \in X :\ x = y)$$

Any interval including the empty interval is set equal to itself.

$$X\text{.SEQ.}X \equiv TRUE$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if $\underline{x} = \underline{y}$ and $\overline{x} = \overline{y}$ or both arguments are empty (4.1). Otherwise the result has the value false.

Algorithm. $\underline{x} = \underline{y}$ and $\overline{x} = \overline{y}$ or $EMPTY(X)$ and $EMPTY(Y)$

11.2 Set inequality. (X .SNE. Y)

Description. Determines the inequality of two intervals as sets.

$$X \text{ .SNE.} Y \equiv (\exists x \in X,\ \forall y \in Y :\ x \neq y) \text{ or } (\exists y \in Y, \forall x \in X :\ x \neq y)$$

Any interval including the empty interval is set equal to itself, therefore

$$X\text{.SNE.}X \equiv FALSE.$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} \neq \underline{y}$ or $\overline{x} \neq \overline{y}$) or either argument is the empty interval. Otherwise the result has the value false.

Algorithm.

not ($\underline{x} = \underline{y}$ and $\overline{x} = \overline{y}$ or $EMPTY(X)$ and $EMPTY(Y)$)

Note.

In contrast to the algorithm in [4] the algorithm of the .SNE. operator must use the compliment of the .SEQ. relation (11.1) and must not be implemented comparing the bounds of interval operands as in ($\underline{x} \neq \underline{y}$ or $\overline{x} \neq \overline{y}$). Otherwise we will not get the desired false result if both operands are empty because NaN is unequal to any number including itself.

11.3 Less as set (X.SLT.Y)^{New}

Description.

$$X \text{ .SLT.} Y \equiv (\forall x \in X,\ \exists y \in Y :\ x < y) \text{ and } (\forall y \in Y, \exists x \in X :\ x < y)$$

$$X\text{.SLT.}X \equiv FALSE$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} < \underline{y}$ and $\overline{x} < \overline{y}$) and otherwise it has the value false. The result has the value false if one or both of the arguments is empty (4.1)

Algorithm. $\underline{x} < \underline{y}$ and $\overline{x} < \overline{y}$

11.4 Less-or-equal as set (X.SLE.Y)^{New}

Description.

$$X \text{ .SLE.} Y \equiv (\forall x \in X,\ \exists y \in Y :\ x \leq y) \text{ and } (\forall y \in Y,\ \exists x \in X :\ x \leq y)$$

$$X\text{.SLE.}X \equiv TRUE$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} \leq \underline{y}$ and $\overline{x} \leq \overline{y}$) otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1)

Algorithm. $\underline{x} \leq \underline{y}$ and $\overline{x} \leq \overline{y}$

11.5 Greater as set (X.SGT.Y)^{New}

Description.

$$X \text{ .SGT.} Y \equiv (\forall x \in X,\ \exists y \in Y : x > y \text{ and } (\forall y \in Y, \exists x \in X : x > y)$$

$$X.\text{SGT.}X \equiv FALSE$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} > \underline{y}$ and $\overline{x} > \overline{y}$) otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1)

Algorithm. $\underline{x} > \underline{y}$ and $\overline{x} > \overline{y}$

11.6 Greater-or-equal as set (X.SGE.Y)^{New}

Description.

$$X \text{ .SGE.} Y \equiv (\forall x \in X,\ \exists y \in Y :\ x \geq y) \text{ and } (\forall y \in Y, \exists x \in X :\ x \geq y)$$

$$X.\text{SGE.}X \equiv TRUE$$

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} \geq \underline{y}$ and $\overline{x} \geq \overline{y}$) otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm. $\underline{x} \geq \underline{y}$ and $\overline{x} \geq \overline{y}$

12 Certainly relations

For a relation $\textcircled{\rho} \in \{<, >, \leq, \geq, =, \neq\}$ defined for real numbers the corresponding *certainly* relation for intervals is defined as follows:

$$X \textcircled{\rho}\ Y = TRUE \text{ iff } (\forall x \in X,\ \forall y \in Y :\ x \textcircled{\rho}\ y = TRUE)$$

According to this definition $X \textcircled{\rho}\ X$ is defined in the following way:

$$X \textcircled{\rho}\ X = TRUE \text{ iff } (\forall x' \in X,\ \forall x'' \in X :\ x' \textcircled{\rho}\ x'' = TRUE)$$

therefore for non–degenerate intervals

$$X \textcircled{\rho}\ X = FALSE$$

and compile–time optimization of the expressions X.CLE.X, X.CGE.X and X.CEQ.X must be inhibited.

Certainly relations should be the default ones:

$<$	$>$	$\leq$	$\geq$	$=$	$\neq$
.CLT.	.CGT.	.CLE.	.CGE.	.CEQ.	.CNE.

If the empty interval is an operand of a *certainly* relation then the result is false. The one exception is the *certainly not equal* relation (12.6) which is true in this case.

12.1 Certainly less (X.CLT.Y)

Description.

$$X\ <\ Y\ \equiv\ (\forall x \in X,\ \forall y \in Y :\ x < y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if $(\overline{x} < \underline{y})$ otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm. $\overline{x} < \underline{y}$

12.2 Certainly less-or-equal (X.CLE.Y)

Description.

$$X\ \leq\ Y\ \equiv\ (\forall x \in X,\ \forall y \in Y :\ x \leq y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\overline{x} \leq \underline{y}$) otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm. $\overline{x} \leq \underline{y}$

12.3 Certainly greater(X.CGT.Y)

Description.

$$X \ > \ Y \ \equiv \ (\forall x \in X, \ \forall y \in Y : \ x > y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} > \overline{y}$) otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm. $\underline{x} > \overline{y}$

12.4 Certainly greater-or-equal (X.CGE.Y)

Description.

$$X \ \geq \ Y \ \equiv \ (\forall x \in X, \ \forall y \in Y : \ x \geq y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} \geq \overline{y}$) otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm. $\underline{x} \geq \overline{y}$

12.5 Certainly equal (X.CEQ.Y)

Description.

$$X \ = \ Y \ \equiv \ (\forall x \in X, \ \forall y \in Y : \ x = y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\overline{x} \leq \underline{y}$ and $\underline{x} \geq \overline{y}$) otherwise has the value false. The result has the value false if one of the arguments is empty (4.1)

Algorithm. $\overline{x} \leq \underline{y}$ and $\underline{x} \geq \overline{y}$

12.6 Certainly not-equal (X.CNE.Y)

Description.

$$X \neq Y \equiv (\forall x \in X,\ \forall y \in Y :\ x \neq y)$$

The result is true if one or both of the arguments is empty (4.1)

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} > \overline{y}$ or $\underline{y} > \overline{x}$) otherwise has the value false. The result has the value true if one or both of the arguments is empty (4.1)

Algorithm. not ($\underline{x} \leq \overline{y}$ and $\underline{y} \leq \overline{x}$)

Note.

> *The semantic and the algorithm for the .CNE. operator are equivalent to those of the disjoint operator .DJ.*
>
> *To get the desired true result if one or both operands are empty it is necessary to use the compliment of ($\underline{x} \leq \overline{y}$ and $\underline{y} \leq \overline{x}$) rather than ($\underline{x} > \overline{y}$ or $\underline{y} > \overline{x}$).*

13 Possibly relations

For a relation $\textcircled{\rho} \in \{<, >, \leq, \geq, =, \neq\}$ defined for real numbers the corresponding *possibly* relation $?\textcircled{\rho}$ for intervals is defined as follows:

$$X?\textcircled{\rho}\ Y = TRUE \text{ iff } (\exists x \in X,\ \exists y \in Y :\ x\textcircled{\rho}\ y = TRUE)$$

According to this definition $X?\textcircled{\rho}\ X$ is defined in the following way:

$$X?\textcircled{\rho}\ X = TRUE \text{ iff } (\exists x' \in X,\ \exists x'' \in X :\ x'\textcircled{\rho}\ x'' = TRUE)$$

therefore for non–degenerate intervals

$$X?\textcircled{\rho}\ X = TRUE$$

and compile–time optimization of the expressions X.PLT.X, X.PGT.X and X.PNE.X must be inhibited.

If the empty interval is an operand of a *possibly* relation then the result is false. The one exception is the *possibly not equal* relation (13.6) which is true in this case.

13.1 Possibly less (X.PLT.Y)

Description.

$$X\ .\text{PLT}.Y\ \equiv\ (\exists x \in X,\ \exists y \in Y :\ x < y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if $(\underline{x} < \overline{y})$ otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm. $\underline{x} < \overline{y}$

13.2 Possibly less-or-equal (X.PLE.Y)

Description.

$$X\ .\text{PLE}.Y\ \equiv\ (\exists x \in X,\ \exists y \in Y :\ x \leq y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if $(\underline{x} \leq \overline{y})$ otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm. $\underline{x} \leq \overline{y}$

13.3 Possibly greater (X.PGT.Y)

Description.

$$X \text{ .PGT.} Y \equiv (\exists x \in X,\ \exists y \in Y :\ x > y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if $(\overline{x} > \underline{y})$ otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm. $\overline{x} > \underline{y}$

13.4 Possibly greater-or-equal (X.PGE.Y)

Description.

$$X \text{ .PGE.} Y \equiv (\exists x \in X,\ \exists y \in Y :\ x \geq y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if $(\overline{x} \geq \underline{y})$ otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm.$\overline{x} \geq \underline{y}$

13.5 Possibly equal (X.PEQ.Y)^{New}

Description.

$$X \text{ .PEQ.} Y \;\equiv\; (\exists x \in X,\ \exists y \in Y :\ x = y)$$

The result is false if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\underline{x} \leq \overline{y}$ and $\overline{x} \geq \underline{y}$) otherwise has the value false. The result has the value false if one or both of the arguments is empty (4.1).

Algorithm. $\underline{x} \leq \overline{y}$ and $\overline{x} \geq \underline{y}$

13.6 Possibly not-equal (X.PNE.Y)^{New}

Description.

$$X \text{ .PNE.} Y \;\equiv\; (\exists x \in X,\ \exists y \in Y :\ x \neq y)$$

The result is true if one or both of the arguments is empty (4.1).

Class. Relational interval operation.

Arguments. X and Y shall be of type interval

Result characteristics. Default logical scalar.

Result value. The result has the value true if ($\overline{x} > \underline{y}$ or $\underline{x} < \overline{y}$) otherwise has the value false. The result has the value true if one or both of the arguments is empty (4.1).

Algorithm. not ($\overline{x} \leq \underline{y}$ and $\underline{x} \geq \overline{y}$)

Note.

> *To get the desired TRUE result if one or both operands are empty it is necessary to use the compliment of the algorithm for the* certainly equal *relation rather than ($\overline{x} > \underline{y}$ or $\underline{x} < \overline{y}$).*

14 Special interval functions

14.1 WID(X)

Description. Width of the interval.

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. The result is of type real. The kind type parameter is that of X.

Result value. The result is a processor–dependent approximation of $(\overline{x} - \underline{x})$. If it is not exact then it shall be upwardly rounded.

The result is a quiet NaN if the argument is empty (4.1).

Algorithm.$(\overline{x} - \underline{x})$ rounded up.

14.2 MID(X)

Description. Midpoint of the interval.

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. The result is of type real. The kind type parameter is that of X.

Result value. The result is a processor–dependent approximation of $(\underline{x} + \overline{x})/2$. If it is not exact then it shall be rounded to the nearest.

The result is a quiet NaN if the argument is empty (4.1).

Algorithm. $(\underline{x} + \overline{x})/2$

14.3 MIG(X)

Description. The smallest absolute value in the interval. $\min\{|x| \mid x \in X\}$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. The result is of type real. The kind type parameter is that of X.

Result value. The result is $\min\{|\underline{x}|, |\overline{x}|\}$, if $0 \notin X$; otherwise the result is 0.

The result is a quiet NaN if the argument is empty (4.1).

Algorithm.

$$\begin{array}{lll} \min\{|\underline{x}|, |\overline{x}|\} & : & \text{if } 0 \notin X \\ 0 & : & \text{otherwise.} \end{array}$$

14.4 MAG(X)

Description. The greatest absolute value in the interval. $\max\{|x| \mid x \in X\}$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. The result is of type real. The kind type parameter is that of X.

Result value. $\max\{|\underline{x}|, |\overline{x}|\}$.

The result is a quiet NaN if the argument is empty (4.1).

Algorithm. $\max\{|\underline{x}|, |\overline{x}|\}$

14.5 ABS(X)

Description. Range of absolute value.

$$ABS(X) \supseteq \{|x| \mid x \in X\} \equiv [MIG(X), MAG(X)]$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

If if the argument is empty (4.1), the result is the empty interval.

Algorithm. $[MIG(X), MAG(X)]$

14.6 MAX(X,Y)

Description. Range of maximum.

$$MAX(X,Y) \supseteq \{\max(x,y) \mid x \in X,\ y \in Y\} \equiv [\max\{\underline{x},\underline{y}\}, \max\{\overline{x},\overline{y}\}]$$

Class. Elemental function.

Arguments. The arguments shall be of type interval and all have the same kind type parameter.

Result characteristics. Same as the arguments.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

If the argument is empty (4.1), the result is the empty interval.

Algorithm. $[\max\{\underline{x},\underline{y}\}, \max\{\overline{x},\overline{y}\}]$

14.7 MIN(X,Y)

Description. Range of minimum.

$$MIN(X,Y) \supseteq \{\min(x,y) \mid x \in X,\ y \in Y\} \equiv [\min\{\underline{x},\underline{y}\}, \min\{\overline{x},\overline{y}\}]$$

Class. Elemental function.

Arguments. The arguments shall be of type interval and all have the same kind type parameter.

Result characteristics. Same as the arguments.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

If the argument is empty (4.1), the result is the empty interval.

Algorithm. $[\min\{\underline{x},\underline{y}\}, \min\{\overline{x},\overline{y}\}]$

14.8 NDIGITS(X)

Description. Maximum number of decimal digits that will be displayed printing X with the SE or SF interval editing descriptor (see 19.3).

Class. Inquiry function.

Arguments. X shall be of type interval

Result characteristics. Default integer scalar. If the argument is empty (4.1), the result is zero.

Result value. Maximum number of decimal digits in a number y shall be such that X is contained in an interval whos bounds are defined by adding and subtracting 1 to the last decimal digit of y. See 18.2.

If X is a degenerate interval ($\underline{x} = \overline{x}$), then the result is MAX_INT.

Examples NDIGITS ($[0.0, .2]$) = 1
NDIGITS ($[1, 1]$) = MAX_INT
NDIGITS ($[2.345677, 2.345679]$) = 7

15 Real–valued intrinsic functions with interval arguments

15.1 AINT(X, [,KIND]

Description. Truncation of the midpoint of the interval argument to a whole number.

Class. Elemental function.

Arguments.

X shall be of type interval.

KIND (optional) shall be a scalar integer initialization expression.

Result characteristics. If KIND is present, the kind type parameter is that specified by KIND; otherwise the kind type parameter is that of X.

Result value. AINT(MID(X))

15.2 ANINT(X, [,KIND]

Description. The whole number nearest to the midpoint of the interval argument.

Class. Elemental function.

Arguments.

X shall be of type interval.

KIND (optional) shall be a scalar integer initialization expression.

Result characteristics. If KIND is present, the kind type parameter is that specified by KIND; otherwise the kind type parameter is that of X.

Result value. ANINT(MID(X))

16 Integer–valued intrinsic functions with interval arguments

16.1 INT(X [, KIND])

Description. Convert the midpoint of the argument to integer type.

Class. Elemental function.

Arguments.

X shall be of type interval

KIND (optional) shall be a scalar integer initialization expression.

Result characteristics. If KIND is present, the kind type parameter is that specified by KIND; otherwise the kind type parameter is that of default integer type.

Result value. The result value is INT(MID (X)) .

If the argument is empty the result is a processor dependent value same as the value of the integer-valued INT function with NaN as the argument.

16.2 FLOOR(X [, KIND])

Description. Returns the greatest integer less than or equal to the infimum of its argument.

Class. Elemental function.

Arguments.

X shall be of type interval

KIND (optional) shall be a scalar integer initialization expression.

Result characteristics. If KIND is present, the kind type parameter is that specified by KIND; otherwise the kind type parameter is that of default integer type.

Result value. The result has a value equal to the greatest integer less than or equal to INF(x): FLOOR(INF(X))

If the argument is empty the result is a processor dependent value same as the value of the integer-valued FLOOR function with NaN as the argument.

16.3 CEILING(X [, KIND])

Description. Returns the least integer greater than or equal to the supremum of its argument.

Class. Elemental function.

Arguments.

X shall be of type interval

KIND (optional) shall be a scalar integer initialization expression.

Result characteristics. If KIND is present, the kind type parameter is that specified by KIND; otherwise the kind type parameter is that of default integer type.

Result value. The result has a value equal to the least integer greater than or equal to SUP(X): CEILING(SUP(X))

If the argument is empty the result is a processor dependent value same as the value of the integer-valued CEILING function with NaN as the argument.

17 Interval versions of mathematical functions

All Fortran intrinsic functions that accept real data shall also accept interval data.

In Fortran 90 those generic intrinsics that are real elemental functions shall also operate as elemental functions with interval vector data.

Let $\varphi : D \subset \mathbb{IR} \to \mathbb{IR}$ denote a real-valued *intrinsic function*. We extend φ to an interval argument X by

$$\varphi(X) \supseteq \{\varphi(x) \mid x \in X \cap D\}.$$

That is, $\varphi(X)$ denotes the range of the real-valued function φ over X.

By definition, an interval intrinsic function is inclusion monotonic, i.e. $X \subseteq Y \Rightarrow \varphi(X) \subseteq \varphi(Y)$.

If the value of the halting mode INTERVAL_POSSIBLY_OUT_OF_RANGE is false, then the argument of the intrinsic is intersected with the valid domain of the interval intrinsic. (The valid domain of an interval intrinsic function shall be the set of

intervals contained in the valid domain of the corresponding floating–point intrinsic function.) The result is empty if the intersection is empty. See also 5.2 and 5.8.

17.1 SQRT(X)

Description. Square root of the interval.

$$\sqrt{X} \supseteq \{\sqrt{x} \mid x \in X \cap [0, \infty)\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.2 EXP(X)

Description. Exponential of the interval.

$$\exp(X) \supseteq \{\exp(x) \mid x \in X\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.3 LOG(X)

Description. Interval natural logarithm.

$$\log(X) \supseteq \{\log(x) \mid x \in X \cap [0, +\infty)\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.4 LOG10(X)

Description. Interval common logarithm.

$$\log 10(X) \supseteq \ \{\log 10(x) \mid x \in X \cap \ [0, +\infty)\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.5 MOD(X,Y)

Description.

$$\mathrm{mod}(X, Y) \supseteq \ \{\mathrm{mod}(x, y) \mid x \in X, y \in Y\}.$$

Class. Elemental function.

Arguments. The arguments shall be of type interval and of the same kind type parameter.

Result characteristics. Same as the arguments.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if one or both arguments are empty.

17.6 SIGN(X,Y)

Description.

$$\text{sign}(X, Y) \supseteq \{\text{sign}(x, y) \mid x \in X, y \in Y\}.$$

Class. Elemental function.

Arguments. The arguments shall be of type interval and of the same kind type parameter.

Result characteristics. Same as the arguments.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if one or both arguments are empty.

17.7 SIN(X)

Description. Interval sine function.

$$\sin(X) \supseteq \{\sin(x) \mid x \in X\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.8 COS(X)

Description. Interval cosine function.

$$\cos(X) \supseteq \{\cos(x) \mid x \in X\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.9 TAN(X)

Description. Interval Tangent function.

$$\tan(X) \supseteq \ \{\tan(x) \mid x \in X\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.10 ASIN(X)

Description. Interval inverse sine function.

$$\text{asin}(X) \supseteq \ \{\text{asin}(x) \mid x \in X \cap \ [-1, 1]\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.11 ACOS(X)

Description. Interval Inverse cosine function.

$$\text{acos}(X) \supseteq \ \{\text{acos}(x) \mid x \in X \cap \ [-1, 1]\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.12 ATAN(X)

Description. Interval inverse tangent function.

$$\text{atan}(X) \supseteq \{\text{atan}(x) \mid x \in X\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.13 ATAN2(X,Y)

Description. Interval inverse tangent function.

$$\text{atan2}(X,Y) \supseteq \{\text{atan2}(x) \mid x \in X,\ y \in Y, 0 \notin X \cap\ Y\}.$$

Class. Elemental function.

Arguments. X shall be of type interval, Y shall be of the same type and kind type parameter as X.

Result characteristics. Same as the arguments.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if one or both arguments are empty.

17.14 SINH(X)

Description. Interval hyperbloic sine function

$$\sinh(X) \supseteq \{\sinh(x) \mid x \in X\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.15 COSH(X)

Description. Interval hyperbloic cosine function.

$$\cosh(X) \supseteq \{\cosh(x) \mid x \in X\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

17.16 TANH(X)

Description. Interval hyperbloic tangent function.

$$\tanh(X) \supseteq \{\tanh(x) \mid x \in X\}.$$

Class. Elemental function.

Arguments. X shall be of type interval

Result characteristics. Same as the argument.

Result value. The interval result value shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

The result is empty if argument is empty.

18 Explicite conversions the interval

18.1 INTERVAL(X,[Y,KIND])

Explicite conversion to the interval from integers, reals or intervals of different kinds shall be performed with the constructor INTERVAL.

Description. Convert to interval type.

Class. Elemental function.

Arguments.

X	shall be of type integer, real or interval.
Y (optional)	shall be of type integer, real or interval. If X is of type interval, Y shall not be present, nor shall be Y associated with an optional dummy argument. If X is of type integer or real, Y shall be larger than or equal to X.
KIND (optional)	shall be a scalar integer initialization expression.

Result characteristics. The result is of type interval. If KIND is present, the kind type parameter is that specified by KIND; If KIND is not present and X is of type interval, the kind type parameter is that of X; otherwise, the kind type parameter is determined by the types of the arguments according to rules for Fortran generic intrinsics (kind type parameter – corresponding to highest precision of arguments), but it shall be at least kind type parameter of the REAL *8 type.

Result value. The interval result value shall be a valid floating-point interval (see section 4) and shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification. When the argument(s) of INTERVAL are literal constants or constant expressions, the specification of INTERVAL shall be the same as if "[" and "]" were used. The result can be only a valid floating-point interval.

Example

INTERVAL(1.0,2.0) results in the interval $[1.0, 2.0]$
INTERVAL(1.0 / 0.0) results in the interval $[MAX\_REAL, +\infty]$

18.2 CONVERT_DECIMAL_DIGITS(STRING, NDIGITS [, KIND])

Description. Construct interval from a single number in the same way as the SE and SF edit descriptors (see 19.3).

Class. Elemental function.

Arguments.

STRING shall be of the type default character.

NDIGITS shall be of type default integer and be non–negative.

KIND (optional) shall be a scalar integer initialization expression.

Result characteristics. Interval. If KIND is present, the kind type parameter is that specified by KIND; otherwise the kind type parameter is the processor–dependent kind type parameter for the default interval type (shall be at least REAL *8 type).

Result value. The interval result value shall be an enclosure for the interval $[\underline{x}, \overline{x}]$, where $\underline{x}$ and $\overline{x}$ are constructed from STRING by subtracting and adding 1 to the NDIGITS digit of STRING respectively. An ideal enclosure shall be equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

Example. Suppose a kind parameter of 2 corresponds to real and interval data types with approximately 15 decimal digits of precision. Then CONVERT('3.14159',6,2) will be an interval that contains the mathematical interval [3.141585, 3.14195]. CONVERT_DECIMAL_DIGITS('3.14159',20,2) will be a machine interval (kind parameter 2) of non-zero width that contains the decimal constant 3.14159. This is precisely the same result that will be obtained using single number I/O.

18.3 CONVERT_WITHIN_BOUNDS(R,EPS [,KIND])

Description. Convert REAL values with known relative error bounds

Class. Elemental function.

Arguments.

R shall be of type real.

EPS shall be of type real.

KIND (optional) shall be a scalar integer initialization expression.

Result characteristics. If KIND is present, the kind type parameter is that specified by KIND; otherwise the kind type parameter is the processor–dependent kind type parameter for the default interval type (shall be at least REAL *8 type).

Result value. The interval result value shall be an enclosure for the mathematical interval [R*(1-EPS),R*(1+EPS)].

Example. Suppose it is known that the real value R has been computed to within an accuracy of one part in a thousand, and R happens to be equal to 2. Then the call

CONVERT_WITHIN_BOUNDS(R,0.001)

will return an INTERVAL of default kind that contains the mathematical interval $[0.999, 1.001]$.

19 Interval I/O editing

The specification presented here is very close to those presented in [6].

The underlying principle is : the interval result shall contain the exact mathematical result. Specifically:

- On input, the stored interval shall contain the interval represented by the character input string. Leading blanks are not significant.
- On output, the printed interval shall contain the internally represented interval. A zero or positive internal value may be prefixed with a plus sign. A negative value is always prefixed with minus. The zero internal value is never prefixed with a leading minus.
- The empty interval shall be printed as the interval $[NaN, NaN]$ according to the specified format.

The VF, VE, SE, SF, SG and VG interval edit descriptors may be used to specify the input/output of interval data.

The following general rules apply:

- In all interval input fields, blanks between an initial "[" and the first numerical fields, blanks to the left and right of a separating ",", and blanks to the left of a closing "]" shall be ignored.

It shall also be possible to input and output interval data with the E and/or F edit descriptors. In that case, two E or F fields, or one of each, are required for each interval datum, and the lower and upper bounds of the interval datum are treated as usual real data.

Warning:

> In this case the printed interval endpoints may not contain the internal representation.

Intervals may be also be input and output with a "G" edit descriptor. Input of an interval with the G edit descriptor shall be identical to input with the SF edit descriptor. Output of an interval with a "G" edit descriptor shall be identical to output with an "SG" edit descriptor if one or more digits of the internal representation correspond, and shall be identical to output with the "VG" edit descriptor otherwise.

Note:

INTERVAL's can also be input and output with NAMELIST and list-directed I/O.

Output of intervals in list-directed I/O shall be identical to output with a "G" edit descriptor, with reasonable, processor-dependent values of w, d, and e.

19.1 The interval VF editing

The $VFw.d$ edit descriptor indicates that each of two numeric fields occupies w positions, each fractional part of which consists of d digits.

19.1.1 The interval VF input editing

An input field for a $VFw.d$ edit descriptor is of the form $vf-input-field$, where

$vf-input-field$	is	$f-input-field$
	or	$[f-input-field]$
	or	$[f-input-field_1, f-input-field_2]$
$f-input-field$	is	a valid input field for the $Fw.d$ format, where w and d are the values specified in the $VFw.d$ edit descriptor.

If $vf-input-field$ is of the form $[f-input-field_1, f-input-field_2]$, then $f-input-field_1$ represents the lower bound and the $f-input-field_2$ represents the upper bound. In this case, the lower bound of the internal representation of the variable in the corresponding input item list shall be less than or equal to $f-input-field_1$, and the upper bound of the variable shall be greater than or equal to $f-input-field_2$, regardless of the number of digits in the field and number of digits in the internal representation.

If $vf-input-field$ is of the form $f-input-field$ or $[f-input-field]$, then the internal representation of the corresponding variable in the input item list shall have lower bound that is less than or equal to the value represented by $f-input-field$, and shall have upper bound that is greater than or equal to $f-input-field$, regardless of the number of digits in the field and number of digits in the internal representation.

19.1.2 The interval VF output editing

An output field for a $VFw.d$ edit descriptor is of the form *vf-output-field*, where

$vf-output-field$	is	$[f-output-field_1, f-output-field_2]$
$f-output-field$	is	the output field for the $Fw.d$ format, where w and d are the values specified in the $VFw.d$ edit descriptor.

The value corresponding to $f-output-field_1$ shall be less than or equal to the exact lower bound of the corresponding output list item, regardless of the number of digits in the field and number of digits in the internal representation. The value corresponding to $f-output-field_2$ shall be greater than or equal to the exact upper bound of the corresponding output list item, regardless of the number of digits in the field and number of digits in the internal representation.

It shall be possible to use the VF edit descriptor to output real data. In that case, the value corresponding to $f-output-field_1$ shall be less than or equal to the exact value of the corresponding real output list item, regardless of the number of digits in the field and number of digits in the internal representation. The value of $f-output-field_2$ shall be greater than or equal to the exact upper bound of the corresponding real output list item, regardless of the number of digits in the field and number of digits in the internal representation.

For both input and output, the symbols "[" , "]" and "," that are part of the field shall not be counted as part of the width w of the overall VF field. The total width of the field is thus $2w+3$.

Examples. Suppose it is required to input the degenerate interval $[1.5, 1.5]$ and the READ statement is:

```
INTERVAL X
READ(*,'(1X,VF18.5)') X
```

Then any of the following input lines result in an internal representation for X that is equal to or contains the interval $[1.5, 1.5]$.

```
1.5
1.5E0
[1.5]
[1.5,1.5]
```

19.2 The interval VE editing

VE editing is analogous to VF editing, except that it corresponds to the E, rather than F, edit descriptor.

The form and interpretation of the *ve-input-field* shall be the same as for VF editing 19.1.1.

An output field for a $VEw.d[Ee]$ edit descriptor shall be of the form *ve-output-field*, where

ve-output-field	is	[*e-output-field*$_1$, *e-output-field*$_2$]
e-output-field	is	the output field for the $Ew.d[Ee]$ format, where w, d, and e are the values specified in the $VEw.d[Ee]$ edit descriptor.

The value corresponding to *e-output-field*$_1$ shall be less than or equal to the exact lower bound of the corresponding output list item, and the value corresponding to *e-output-field*$_2$ shall be greater than or equal to the exact upper bound of the corresponding output list item, regardless of the number of digits in the field and number of digits in the internal representation.

It shall be possible to use the VE edit descriptor to output real data. In that case, the value corresponding to *e-output-field*$_1$ shall be less than or equal to the exact value of the corresponding real output list item, and the value of *e-output-field*$_2$ shall be greater than or equal to the exact upper bound of the corresponding real output list item, regardless of the number of digits in the field and number of digits in the internal representation.

For both input and output, the symbols "[" , "]" and "," that are part of the field shall not be counted as part of the width w of the overall VE field. The total width of the field is thus $2w + 3$.

Example: Suppose an interval variable X is defined in a program, suppose the program contained the statement

WRITE(*,'(1X,VE12.5E1)') X

and suppose the internal representation of X is that of the interval

$$[1.9921875, 2.9921875]$$

Then a valid output produced by the WRITE statement is
`[ 0.19921E+1, 0.29922E+1]`
or
`[ 0.19921E+1, 0.29922E+1]`

19.3 The interval SE and SF editing

The SE and SF formats are for single number input and output of interval's with implied bounds of plus or minus 1 unit in the last exhibited digit.

Due to possible internal base conversions the output with SE and SF formats may display less number of decimal digits than has been inputed with the corresponding format. In order to display the same number of decimal digits in this case character input/output and internal conversion of character strings to intervals must be used. See the function CONVERT_DECIMAL_DIGITS (18.2).

The basic representation of an interval as a single number is as follows:

- A single number without a decimal point shall represent an interval whose lower and upper endpoints are identical (a degenerate, i.e. a point interval).
- A single number with a decimal point shall represent an interval whose endpoints are constructed by subtracting and adding 1 unit to the last exhibited decimal digit.

Examples.

Single Number	Interval represented
`[.1]`	$[0.0, .2]$
`[.15`	$[.14, .16]$
`[0.E3]`	$[-1.E3, +1.E3]$
`[1.E3]`	$[0, 2.E3]$
`[2.345678]`	$[2.345677, 2.345679]$
`[1]`	$[1, 1]$

Note

Degenerate interval decimal constants, such as $[0.1, 0.1]$ or $1E-1$, are not always representable exactly on binary machines. Conversion of such degenerate intervals to internal format is specified by outward rounding, as explained below.

The SE and SF specifiers have the same form as the E and F specifiers, i.e. $SFw.d$ and $SEw.d[Ee]$. However, the output of an SE or SF specifier shall be of the form

[*single-number-output*]

where single-number-output is of the form specified by $Fw.d$ for the $SFw.d$ specifier, and of the form specified by $Ew.d[Ee]$ for the $SEw.d[Ee]$ specifier, with the following differences:

- The number of printed decimal digits in a number y shall be such that X is contained in an interval whos bounds are defined by adding and subtracting 1 to the last decimal digit of y. See 18.2. (See also NDIGITS 14.8 and 18.2)
- For zero-width intervals, neither a decimal point nor digits past the decimal point shall be displayed.
- For the SF specifier, the positions defined in the descriptor that correspond to digits that are not displayed shall be filled by blanks, so the displayed digits are justified as though all digits prescribed by the specifier were printed. For the SE specifier, if only s digits are printed, the output will be in the form of an $Ew.s$ specifier, left-justified in the field that it would occupy if s were equal to d.
- If a number is too inaccurate to be represented within a specified SF format, the entire field will be filled with asterisks.

Note:

> The SF format, when used for zero-width intervals, is limited to integers, since all other zero-width intervals will be output as fields of asterisks.

Note:

> The w specifies the width of the numerical field, and not the spaces taken by "[" and "]". Thus, the total width of an SE or SF field is $w + 2$.

The input for an SE or SF specifier shall be of the form *sf-input-field*, where:

sf-input-field	is	*f-input-field*
	or	[*f-input-field*]
f-input-field	is	a valid input field for the $Fw.d$ format, where w and d are the values specified in the $SFw.d$ or $SEw.d[Ee]$ edit descriptor.

Upon input, the internal representation shall depend upon whether the input string contains a decimal point. If the input string contains a decimal point, the number shall be equal to or contain the interval centered upon the number represented by *f-input-field*, and with width equal to one decimal unit in the least significant digit exhibited. If the input does not contain a decimal point, the internal representation shall be the same as if a VF format specifier is used.

Examples. Suppose the number 1.5 is known to be correct to the last digit represented, to within rounding. Suppose the READ statement is:

```
INTERVAL X
READ(*,'(1X,SF18.5)') X
```

Then any of the following input lines result in an internal representation for X that is equal to or contains the interval $[1.4, 1.6]$.

```
1.5
```
or
```
1.5E0
```
or
```
[1.5]
```

However, the following inputs merely result in an internal representation for X that is equal to or contains the degenerate interval $[1.5, 1.5]$.

```
15E-1
```
or
```
[15E-1]
```

In a second example, inputs of

```
0.E1
```
or
```
[0.E1]
```
or
```
0.0E2
```
or
```
[0.0E2]
```

result in an internal representation for X that contained the interval $[-1.E1, 1.E1]$.

Note:

> In single number interval I/O, input immediately followed by output can appear to suggest that a decimal digit of accuracy has been lost, when in fact radix conversion has caused a 1 ulp increase in the width of the interval stored in the machine. For example, an input of 0.100 followed by an immediate print will result in 0.10. Users and implementers need to expect this behavior.

19.4 The interval SG editing

The interval SG edit descriptor is for general single-number interval input and output.

For input, the SG edit descriptor is identical to the SF edit descriptor.

The form of the interval SG descriptor is $SGw.d$ or $SGw.dEe$, where w is the width of the field and d is the maximum number of digits actually displayed. The method of representation of the output field depends both on the magnitude of the datum being edited and on the number of digits that are equal in the lower bound and upper bound. Define the following:

> Round a lower and upper bound to s significant decimal digits. Then if all s digits agree, the s digits are said to c*correspond.*
>
> Let r be the minimum of d and q, where q is less than or equal to the maximum number of significant digits of the lower and upper bounds of the internal representation that correspond.

If at least one decimal digit of the lower bound and upper bound correspond then the number printed by the $SGw.d$ or $SGw.dEe$ formats shall be the same as that printed by the respective $Gw.r$ and $Gw.rEe$ formats. The printed number is preceded by the string "[" and is followed by the string "]".

In determining the number of digits that correspond, the lower and upper bounds of the internal representation may first be converted to a decimal number in such a way that the converted lower bound is less than or equal to the lower bound of the internal representation and the converted upper bound is greater than or equal to the upper bound of the internal representation. In any case, the number actually printed shall have r digits that are equal to the r most significant decimal digits of the lower bound and upper bound of the internal representation.

Note:

> Ideally, the number of digits determined to correspond should be the number of digits that actually are equal (in the sense of rounding in the last digit) in the internal representation.

If no digits correspond in the above sense, then the output is the same as with an $SEw.0$ or $SEw.0Ee$ edit descriptor, left-justified in the field.

Example. If the internal representation equals $[1, 100]$, then an $SG12.5E1$ edit descriptor can result in output of the form

```
[0.E3        ]
```

This output is interpreted as the interval $[-1000, 1000]$.

19.5 The interval VG editing

The interval VG edit descriptor is identical to the SF edit descriptor for input. For output with $VGd.w$ or $VGd.wEe$, two decimal numbers are printed, preceded by "[", separated by ",", and followed by "]". The formats of the lower bound and upper bound are determined in accordance with the rules for $Gd.w$ or $Gd.w.Ee$ editing respectively, as if the lower bound and upper bound were real output. However, the printed lower bound shall be less than or equal to the lower bound of the internal representation, and the printed upper bound shall be greater than or equal to the upper bound of the internal representation.

20 Acknowledgements

The authors are very thankful to Michael Schulte and Douglas Priest for valuable remarks.

References

[1] R.P. Corbett, *Enhanced Arithmetic for Fortran*, SIGPLAN Notices, Vol. 17, No. 12, 1982

[2] ANSI/IEEE 754-1985 Standard for Binary Floating-Point Arithmetic, Institute of Electrical and Electronics Engineers, New York, 1985.

[3] ISO/IEC JTC1/SC22/WG5 - N1231, "Technical Report for Floating Point Exception Handling.

[4] ANSI X3J3 1996-156, Interval Arithmetic—The Data Type and Low-Level Operations, 1996.

[5] X3J3/97-155, ISO/IEC JTC1/SC22/WG5 - N1231, Proposed Syntax – Exceptions for Interval Intrinsic Functions.

[6] R. B. Kearfott et al. A specific proposal for interval arithmetic in Fortran, 1996.

[7] M77 Reference Manual. Minnesota Fortran 1977 Standards Version Edition 1. University of Minnesota, 1983.

[8] Moore R.E. *"Methods and applications of interval analysis"* , SIAM Studies in Applied Mathematics. SIAM, Philadelphia, Pennsylvania, 1979.

[9] Popova, E., *Interval Operations Involving NaNs*, Reliable Compuing Vol. 2, No. 2, pp. 161–165.

[10] Priest D., *Handling IEEE 754 Invalid Operation Exceptions in Real Interval Arithmetic*, Manuscript, 1997.

[11] G.W. Walster, *Philosophy and Practicalities of Interval Arithmetic*, in Moore (Ed.) Reliability in Computing. Academic Press, Inc., San Diego, California, pp. 309–323, 1988.

[12] G.W. Walster and E.R. Hansen, *Composite Functions in Interval Mathematics*, Manuscript, 1997. Available as J3/97-198 document.

[13] A.G. Yakovlev, *Classification approach to programming of localizational (interval) computations*, Interval Computations, **1**(3), 1992.