Journal of Universal Computer Science, vol. 3, no. 1 (1997), 23-41
submitted: 16/6/96, accepted: 25/9/96, appeared: 28/1/97 © Springer Pub. Co.

Generalizing BIAS Specifications

Evgenija D. Popova
(Bulgarian Academy of Sciences, Bulgaria
epopova@bgearn.acad.bg)

Christian P. Ullrich
(University of Basel, Switzerland
ullrich@ifi.unibas.ch)

Abstract: This paper generalizes the specification of Basic Interval Arithmetic Subroutines (BIAS) to support interval arithmetic on directed (i.e. proper and improper) intervals. This is due to our understanding that the arithmetic involving improper intervals will be increasingly used in future applications and the corresponding interval arithmetic implementations require no additional cost. We extend BIAS specification to be sufficiently precise and complete, to include everything a user needs, such as subroutine's purpose, name, method of invocation and details of its behaviour and communication with the environment. The specified interval arithmetic subroutines for directed intervals are consistent with conventional interval arithmetic and IEEE floating-point arithmetic.

Key Words: specification, interval arithmetic

Category: D.2.1, D.3., K.6.3

1 Introduction

Interval arithmetic [Alefeld and Herzberger 1974], [Moore 1966] is widely recognized as a valuable computing technique. Numerous applications benefit from methods with result verification – a rapidly growing field of numerical mathematics (see for example [Adams and Kulisch 1993]). A constant effort is devoted to develop algorithms which produce sharp inner and outer bounds for the solution of various numerical problems involving interval input data. A straightforward approach on this way is the algebraic extension of the conventional interval arithmetic called here directed interval arithmetic. Developed basically by E. Kaucher [Kaucher 1973]–[Kaucher 1980] and further investigated by E. Gardenes [Gardenes and Trepat 1980] and others [Dimitrova et al. 1992], [Markov 1995], directed interval arithmetic is obtained as an extension of the set of normal intervals by improper intervals and a corresponding extension of the definitions of the interval arithmetic operations. The algebraic completeness of the corresponding extended interval arithmetic structure (especially the existence of inverse elements with respect to addition and multiplication) makes it a suitable environment for embedding some conventional interval problems [Gardenes and Trepat 1980], [Kupriyanova 1995], [Shary 1996a] and their effective solution. It can be exploited for algebraic simplification of interval expressions and automatic theorem proving if implemented in computer algebra systems. Directed interval arithmetic is useful for a straightforward computation of inner and outer inclusion of functional ranges [Gardenes and Trepat 1980], [Markov 1992]. It seems to be promising for the numerical solution of various practical problems related to interpolation and identification under uncertainties, control theory etc. [Gardenes and Trepat 1980], [Markov 1993], [Shary 1996b]. Certain implementations [Gardenes and Trepat 1980], [Nesterov 1995], [Popova 1994] of the arithmetic on directed intervals appeared to facilitate the numerical computations in extended interval spaces and to provide suitable tools for education and future investigations.

Following the principles of abstract data type design [Goguen et al. 1978], one arrives at a clear separation of specification and implementation. This allows the formulation of the intended behaviour of a data type without referring to its implementation. Any implementation can be used later for the actual computation. Moreover, if in applications the specified operator symbols are used exclusively, their implementation can be changed without changing any application code. In order to facilitate future effective implementations of the directed interval arithmetic in different programming environments we have carefully designed rigorous specifications of the interval arithmetic operations and functions involving directed intervals.

The increasing number of numerical methods with result verification and packages supporting interval arithmetic led to the necessity of unification of the interval arithmetic operations and functions. By analogy with the Basic Linear Algebra Subroutines (BLAS) library [Dongara et al. 1985], [Lawson et al. 1979] some proposals for a Basic Interval Arithmetic Subroutines (BIAS) library appeared recently [Corliss 1990], [Knueppel 1993], showing a movement toward standardization of the user interfaces for interval arithmetic software. The BIAS package [Knueppel 1993], [Knueppel 1994] is well along with the layer model of BLAS and provides portability of the application code together with an increased machine dependent efficiency. On the other hand, directed interval arithmetic includes conventional interval arithmetic as a special case. The defining formulae of the arithmetic operations for directed intervals in end-point representation are identical with the defining formulae of the conventional interval arithmetic operations (see for example [Dimitrova et al. 1992], [Markov 1995]) extending just the scope of their validity. This reveals an important property of the end-point representation of directed interval arithmetic — it can be implemented at no additional cost compared to the conventional interval arithmetic. Thus we stated as a first goal of this paper to generalize the specification of scalar Basic Interval Arithmetic Subroutines [Knueppel 1993] to support interval arithmetic on directed intervals. Having generalized specifications of level 0 interval scalar-scalar operations, next levels 1, 2, 3 of BIAS, built on level 0 subroutines, will automatically support directed intervals. In order to distinguish between specifications and subroutines for conventional interval arithmetic and their extension for directed intervals we call latter Generalized Interval Arithmetic Subroutines (GIAS).

Designing GIAS specification we have kept to a specification scheme [Parnas 1972] ensuring completeness (in the sense of defining all possible uses) and correctness of the specification with no exceeding information. Although specified for floating-point arithmetic and implemented in IEEE floating-point environment [ANSI/IEEE 1985], BIAS specification [Knueppel 1993] is not sufficiently complete to allow checking the hierarchical consistency of interval arithmetic specification with the underlying IEEE arithmetic. This paper defines extensions of BIAS specification for interval arithmetic on directed intervals that are consistent with conventional interval arithmetic and IEEE floating-point arithmetic [ANSI/IEEE 1985], [ANSI/IEEE 1987]. The key to IEEE consistency is the specification of the operations on intervals involving NaNs or signed zero which offer additional benefits of more predictable interval results and algorithm efficiency for some numerical computations. Our present specifications result in an effort to extend the scope of the validity of BIAS and to make them more robust.

2 Basic Concepts and Notations

We assume that the reader is familiar with the interval arithmetic on directed intervals. It has been introduced in [Kaucher 1973]–[Kaucher 1980] and further discussed in [Gardenes and Trepat 1980], [Markov 1995], [Popova 1994], and others. Some relations between conventional and directed interval arithmetic are considered in [Dimitrova et al. 1992] and [Markov 1995].

The following data types are used by GIAS:

REAL – real type (may support any floating-point data format)
INT – integer type
INTERVAL – interval type.

We assume that the interval data type supports end-point representation of directed intervals (unlike "center-radius" or "direction-proper interval" representation). The representation "direction-proper interval" is used for projecting directed intervals onto the space of normal intervals [Markov 1995] but all the properties of directed interval arithmetic are studied for the end-point representation. Furthermore, end-point representation of directed intervals provides consistency with conventional interval arithmetic and an implementation based on this representation will be much more efficient than any implementation based on the representation "direction-proper interval". Since the requirement *"first end-point $\leq$ second end-point"* is not imposed on directed intervals, they can be equally well supported by an opaque data type (using data encapsulation concept) or non-opaque data type [Popova 1994].

Present specification provides all the information an implementor or user will need to complete and/or use the subroutines correctly. For each subroutine we have specified:

1. the set of all possible values, parameters, subroutine name and method of invocation;
2. purpose;
3. description containing the outcome of all possible uses;
4. exceptional situations and their default response.

In our proposal to generalize BIAS [Knueppel 1993] for scalar operations on directed intervals we used the similar routine headers as in BIAS. All routine names are starting with Gias instead of Bias. After that, the purpose of the routine is following (e.g. Mul for multiplication). Finally, the main parameter description is appending, where the letter R denotes a real and the letter I – an interval parameter. For example, the addition of a real and an interval value is named GiasAddRI.

A small number of routines with unique mnemonic names are additionally included in GIAS specification to facilitate directed interval arithmetic. Headers of the additional routines are also kept to the naming conventions for the routines of BIAS. `GiasDual()`, `GiasProper()` and `GiasDirection()` are specific for the manipulation of directed intervals. `GiasSign()` plays an important role for checking monotonicity of a function over an interval and is frequently used in numerical algorithms. It also involves some programming effort that is likely to be ignored in the applications programming environment. `GiasFirst()` and `GiasSecond()` are specified to provide access to the corresponding end-point of a directed interval since `GiasInf()` and `GiasSup()` are not yet sufficient for full characterization of the end-points of a directed interval. `GiasEqualRI()`, `GiasEqualIR()` and `GiasEqualII()` are provided to bridge the existing gap in BIAS [Knueppel 1993] and `GiasLessEqRI()`, `GiasLessEqIR()`, `GiasLessEqII()` and `GiasLessRI()`, `GiasLessIR()`, `GiasLessII()` permit testing of another reflexive and antireflexive interval order relation.

Performing operations which are extended to be valid for a wider set of intervals, Generalized Interval Arithmetic Subroutines produce the same results as Basic Interval Arithmetic Subroutines on ordinary (proper) intervals. The only exception is `GiasIntersection()` which produces an improper interval whenever `BiasIntersection()` results in an empty set. We have also used another definition of distance between two intervals when specifying `GiasDistRI()` and `GiasDistII()`.

Throughout GIAS specification the interval parameters are denoted by capital letters and real or integer parameters by small letters. A variable name followed by number 1 or 2 refers to the corresponding end-point of an interval variable (e.g. A1 denotes the first end-point and A2 the second end-point of the interval variable A).

Note that "contains" is used in GIAS specification in context of an extended inclusion relation: $A \subseteq B \Longleftrightarrow B1 \leq A1$ and $A2 \leq B2$. This means e. g. that a point interval $[r, r]$ contains an improper directed interval $[A1, A2]$ if $A2 \leq r \leq A1$.

The "description" section specifies the outcome of all possible uses of the corresponding subroutine. This section refers to the values of subroutine parameters, the two end-points of the interval parameters and values of other subroutines included in the specification. The description of Generalized Interval Arithmetic Subroutines is given in terms of floating-point operations with directed roundings according to the definitions of computer operations on directed intervals considered in [Durst 1975] and [Kaucher 1973]. Symbols $\bigtriangledown, \bigtriangleup$ denote the corresponding monotone downwardly and monotone upwardly directed roundings $\bigtriangledown, \bigtriangleup : R \longrightarrow SR$, where SR is the set of computer representable real numbers [Kulisch and Miranker 1981], [ANSI/IEEE 1985]. Present specification provides complete information about interval operations on operands involving NaNs (Not a Number) and/or signed zero as part of IEEE floating-point systems [ANSI/IEEE 1985], [ANSI/IEEE 1987]. Some interval arithmetic operations involve algorithmic and implementation subtleties with respect to NaNs and signed zero handling. Thus the provision of a detailed specification benefits both the design and coding stages of a programming effort regarding interval operations on directed intervals.

To make this specification consistent with the environment of IEEE floating-point systems we have provided an "exception" section which specifies interval arithmetic exceptions and their handling according to the principle proposed in [Popova 1996]. Since interval operations are compound operations, all the exceptions arising on execution of an interval operation are floating-point exceptions arising on IEEE floating-point operations [ANSI/IEEE 1985], [ANSI/IEEE 1987] which compound the corresponding interval operation. In this paper we do not describe IEEE floating-point standards 754 and 854 but instead refer to these documents implicitly.

3 Implementation

The objective of the interval arithmetic specification is portability of the application code independently of a specific interval representation and an increased machine dependent efficiency. The set of routines defined by this specification can easily be implemented in any programming environment: as a part of a problem-oriented software system, as a subroutine library for some programming language, or using the operator and overloading concept of some high-level object-oriented programming environment.

3.1 Consistency between conventional and directed interval arithmetic

Major advantages of the directed interval arithmetic are its algebraic properties and that it includes conventional interval arithmetic as a special case. Latter implies some important implementation consequences. Directed interval arithmetic can be implemented at no additional cost compared to conventional interval arithmetic. One and the same data type can be used for both conventional and directed intervals. There is no need of separate implementations. An implementation of directed interval arithmetic as specified below can be used for the execution of any application code concerning a conventional interval arithmetic problem. Furthermore, no change in the application code is required if the implementation is based on operator symbols or subroutines with equal mnemonic names. The only confusion may come from the interval intersection because `GiasIntersection` produces an improper interval whenever `BiasIntersection` delivers an empty set. Two alternatives are possible for solving this problem:

- Any improper interval has to be considered by the conventional interval arithmetic as an empty set. The advantage of this approach is that interval exception *Empty Set Intersection* will have a non-stop exception handling. For example, a preliminary draft proposal [Kearfott et al. 1996] for interval arithmetic in Fortran has specified the empty set as the improper interval $[+\infty, -\infty]$.
- The alternative is setting a global variable or calling a subroutine defining the arithmetic mode. Then, on arguments which are disjoint proper intervals `GiasIntersection` will deliver an interval result if the arithmetic mode is "directed" or will signal interval exception *Empty Set Intersection* if the arithmetic mode is "conventional".

Vendors, software developers and end-users should be made also aware about other benefits brought by the implementation of directed interval arithmetic: suitable environment for effective solution of interval algebraic problems, possibilities of tight range computation for rational interval functions, possibilities for obtaining inner inclusions only by means of outwardly rounded arithmetic operations, etc.

3.2 Consistency between IEEE and non-IEEE implementations

Present specification of directed interval arithmetic is especially designed to provide hierarchical consistency between interval and IEEE floating-point arithmetic, since most recent implementations of floating-point arithmetic conform to the IEEE standards [ANSI/IEEE 1985], [ANSI/IEEE 1987]. Although non-IEEE systems do not support many features of the standard, GIAS specification can also be used for non-IEEE implementation of directed interval arithmetic.

Implementations using IEEE directed roundings toward $-\infty$ (downward, denoted by $\bigtriangledown$) and toward $+\infty$ (upward, denoted by $\bigtriangleup$) provide that the result of the interval operations is the machine interval with smallest diameter containing the exact mathematical result. By some extra programming non-IEEE implementations can provide less accurate results but containing the exact mathematical result.

Special elements like NaNs, -0, $-\infty$, and $+\infty$ are specific for IEEE systems. Description and exception parts of GIAS specifications contain complete information on how to handle NaNs and signed zero involved as end-point(s) of an interval argument. NaNs and/or (-0) participate in some logical expressions of the description section defining the outcome of a subroutine. For non-IEEE implementations the value of a logical expression involving NaNs and/or (-0) should be considered as false.

Usually non-IEEE systems do not support symbols for $-\infty$ and $+\infty$. Since GIAS specification does not refer to these elements explicitly, `GiasSign` specification should be read as:

INTEGER GiasSign(INTERVAL A)
Purpose: Returns -1, if A is a negative directed interval; 1, if A is positive and 0, if 0 is contained in the proper part of A.
Description: If $A1 < 0$ and $A2 < 0$ then -1
elseIf $A1 > 0$ and $A2 > 0$ then 1 else 0.
Exceptions: None.

With respect to the exceptional situations a non-IEEE implementation of the directed interval arithmetic should correspond to the IEEE implementation with all traps, but *Underflow* and *Inexact*, enabled. That is, interval operation terminates immediately on *Invalid Operation*, *Overflow*, and *Division by Zero* floating-point exceptions, and takes special actions to provide inclusion of the exact interval result on floating-point *Underflow* and *Inexact* exceptions. Some implementations may provide true inclusion of the exact interval result if floating-point exception *Overflow* occurs on equally signed arguments at first end-point of the result, or on arguments with different signs at second end-point of the interval result. A non-IEEE implementation supporting symbols $-\infty$ and $+\infty$ can deliver correct inclusion for the exact interval result on all occurrences of floating-point exception *Overflow*.

Although present specifications require an interval involving *quiet* NaN(s) to be delivered by every subroutine which argument involves NaN signaling *Invalid Operation* exception is neither required nor prohibited. In IEEE systems NaNs cause signaling of *Invalid Operation* on floating-point comparison operations. Interval subroutines using such comparisons deliver an interval result and, in the spirit of IEEE standard (*Invalid Operation* is raised if NaN is created from non-NaN operands), may avoid signaling of *Invalid Operation* exception.

3.3 Implementation case study

Here we present a GIAS implementation in a high-level programming environment. PASCAL-XSC [Klatte et al. 1992] was chosen as a wide-spread and best developed programming language for scientific computation ([Ullrich 1994]). A PASCAL-XSC module EXI_ARI [Popova 1994] was developed, where GIAS were implemented as intrinsic operators using the extended operator concept and the overloading concept provided by the language. This new PASCAL-XSC module for extended interval arithmetic was intended to replace the existing module I_ARI for interval arithmetic in PASCAL-XSC language enhancing the scope of application and the robustness of the interval arithmetic operations.

The new module uses the definition of the type `INTERVAL`

type interval = **record** inf, sup : real **end;**

which is part of the language kernel of PASCAL-XSC. The `inf` component of the interval data type corresponds to the first end-point of a directed interval and the `sup` component corresponds to the second end-point of the directed interval comprising thus the definition for a real directed interval as an ordered couple of real numbers.

PASCAL-XSC allows the following three ways for implementation of GIAS corresponding to the interval arithmetic, lattice and relational operations:

– hiding an existing operator. In this case, the operator symbol is given a new meaning for the same operand types for which it was previously defined.
– overloading of an existing operator. A new additional meaning is given to the operator symbol by means of an operator definition. Its various meanings are distinguished by the operand types.
– a previously undefined operator symbol can be introduced by means of an operator declaration.

All arithmetic and lattice operators, defined in EXI_ARI module (Table 1), deliver an interval result. The two monadic operators `+, -` and the four basic dyadic operators `+, -, *, /`, performing the corresponding operation in the module I_ARI with rounding to the smallest enclosing interval (outward rounding), are redefined to perform the same operation for directed intervals. The overloading rules of PASCAL-XSC do not allow to use the result type of a function or an operator for identification of the corresponding subroutine. Thus, new operator symbols `AHO`, `SHO`, `MHO`, `DHO` were used for the operations producing a directed interval enclosing the true result of the corresponding operation on real arguments. `GiasNeg()` and `GiasDual()` were implemented in EXI_ARI module by the corresponding unary operators "$-$" and "_" (see Table 1).

Convex hull operations were implemented in EXI_ARI by overloading of the PASCAL-XSC operator "$+*$" for real, interval and mixed type operands. In the set of directed intervals the lattice operation intersection makes sense even for real number and interval, so the operator "$**$" in module EXI_ARI is also defined for mixed type operands. `GiasInteriorRI/IR,II()` were implemented by overloading the PASCAL_XSC operator `in` and `GiasInRI/IR,II()` — by the relational operator "$<=$".

All utility functions implemented in PASCAL–XSC module EXI_ARI are presented in Table 2.

right Operand / left Operand	integer real	interval
unary		$+,\ -,\ \_$
integer real	$\circ$ $+*,\ **$	$\diamond$ $=,\ <>,\ \sqsubseteq,\ \trianglerighteq$ in, $><,\ +*,\ **$
interval	$\diamond$ $=,\ <>,\ \sqsubseteq,\ \trianglerighteq$ $+*,\ **$	$\diamond$ $=,\ <>,\ \sqsubseteq,\ \trianglerighteq$ in, $><,\ +*,\ **$

$\diamond \in \{+, -, *, /, + <, - <, * <, / <\}$, $\circ \in \{$`AHO`, `SHO`, `MHO`, `DHO`,`AHI`, `SHI`, `MHI`, `DHI`$\}$
$\sqsubseteq \in \{<, <=, >, >=\}$, $\trianglerighteq \in \{$`LT`, `LE`, `GT`, `GE`$\}$

Table 1: The operators of PASCAL–XSC module EXI_ARI

Function	Result Type	Meaning
intval (r1, r2)	*interval*	Interval with $inf = r1$ and $sup = r2$
intval (r)	*interval*	Interval with $inf = sup = r$
pro (i)	*interval*	The proper projection of i
inf (i)	*real*	The smaller end-point of i
sup (i)	*real*	The greater end-point of i
first (i)	*real*	The first end-point of i
second (i)	*real*	The second end-point of i
drc(i)	*integer*	Direction
sgn(i)	*integer*	Sign
mid (i)	*real*	Midpoint of i
diam (i)	*real*	Diameter of i
abs (i)	*interval*	Absolute value $\mid i \mid = \{\mid r \mid : r \in i\}$
dist(r,i)	*real*	Distance between a real value and an interval
dist(i1,i2)	*real*	Distance between two intervals

r, r1, r2 = real expression, i, i1, i2 = interval expression

Table 2: The utility functions of PASCAL–XSC module EXI_ARI

Some other operations and functions involving directed intervals were implemented in EXI_ARI module for convenience of the user. For example, interval operations computing an inner inclusion of the true interval solution (inwardly directed rounding) can also be very useful. An inner inclusion interval is a directed interval which is contained (according to the extended inclusion relation) in the true solution interval. The operators `+<, -<, *<, /<` are redefined for interval and mixed type arguments and new operators `AHI`, `SHI`, `MHI`, `DHI` are defined for real arguments to perform the corresponding operation for directed intervals with inward rounding. Relational operators "<>" (not equal), "<", ">=", ">", "LT", "GT", "GE" are overloaded to test additionally the corresponding order relation between directed intervals.

4 Specification of the GIAS Scalar Operations

GiasAddRR(INTERVAL R, REAL a, REAL b)
Purpose: Calculates the directed interval R including the true result of $a + b$.
Description: $R = [\bigtriangledown(a + b),\ \bigtriangleup(a + b)]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point additions. Handling by the corresponding floating-point system according to the IEEE Std.

GiasAddRI(INTERVAL R, REAL a, INTERVAL B)
Purpose: Calculates the directed interval R including the true result of $a + B$.
Description: $R = [\bigtriangledown(a + B1),\ \bigtriangleup(a + B2)]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point additions. Handling by the corresponding floating-point system according to the IEEE Std.

GiasAddIR(INTERVAL R, INTERVAL A, REAL b)
Purpose: Calculates the directed interval R including the true result of $A + b$.
Description: $R = [\bigtriangledown(A1 + b),\ \bigtriangleup(A2 + b)]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point additions. Handling by the corresponding floating-point system according to the IEEE Std.

GiasAddII(INTERVAL R, INTERVAL A, INTERVAL B)
Purpose: Calculates the directed interval R including the true result of $A + B$.
Description: $R = [\bigtriangledown(A1 + B1),\ \bigtriangleup(A2 + B2)]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point additions. Handling by the corresponding floating-point system according to the IEEE Std.

GiasNeg(INTERVAL R, INTERVAL A)
Purpose: Unary minus of a directed interval. Calculates $R = -A$.
Description: $R = [-A2,\ -A1]$.
Exceptions: Depend on the implementation of floating-point copying operation. If it is implemented as an arithmetic operation then all the exceptions are caused by the corresponding floating-point exceptional operands. There will be no exceptions if floating-point copying operation is treated as non-arithmetic operation.

GiasSubRR(INTERVAL R, REAL a, REAL b)
Purpose: Calculates the directed interval R including the true result of $a - b$.
Description: $R = [\bigtriangledown(a - b),\ \bigtriangleup(a - b)]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point subtractions. Handling by the corresponding floating-point system according to the IEEE Std.

GiasSubRI(INTERVAL R, REAL a, INTERVAL B)
Purpose: Calculates the directed interval R including the true result of $a - B$.
Description: $R = [\bigtriangledown(a - B2),\ \bigtriangleup(a - B1)]$.

Exceptions: All caused by exceptional operands and exceptional results on floating-point subtractions. Handling by the corresponding floating-point system according to the IEEE Std.

GiasSubIR(INTERVAL R, INTERVAL A, REAL b)
Purpose: Calculates the directed interval R including the true result of $A - b$.
Description: $R = [\bigtriangledown(A1 - b),\ \triangle(A2 - b)]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point subtractions. Handling by the corresponding floating-point system according to the IEEE Std.

GiasSubII(INTERVAL R, INTERVAL A, INTERVAL B)
Purpose: Calculates the directed interval R including the true result of $A - B$.
Description: $R = [\bigtriangledown(A1 - B2),\ \triangle(A2 - B1)]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point subtractions. Handling by the corresponding floating-point system according to the IEEE Std.

GiasMulRR(INTERVAL R, REAL a, REAL b)
Purpose: Calculates the directed interval R including the true result of $a \times b$.
Description: $R = [\bigtriangledown(a \times b),\ \triangle(a \times b)]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point multiplications. Handling by the corresponding floating-point system according to the IEEE Std.

GiasMulRI(INTERVAL R, REAL a, INTERVAL B)
Purpose: Calculates the directed interval R including the true result of $a \times B$.
Description: $R = \begin{cases} [\bigtriangledown(a \times B2),\ \triangle(a \times B1)], & \text{if } a < 0 \text{ or } a = -0 \\ [\bigtriangledown(a \times B1),\ \triangle(a \times B2)], & \text{otherwise.} \end{cases}$
Exceptions: All caused by exceptional operands and exceptional results on floating-point multiplications. Handling by the corresponding floating-point system according to the IEEE Std. In addition, *Invalid Operation* exception may be signaled if the real operand is a NaN. The delivered default result is [qNaN, qNaN] if the exception occurs without a trap.

GiasMulIR(INTERVAL R, INTERVAL A, REAL b)
Purpose: Calculates the directed interval R including the true result of $A \times b$.
Description: $R = \begin{cases} [\bigtriangledown(A2 \times b),\ \triangle(A1 \times b)], & \text{if } b < 0 \text{ or } b = -0 \\ [\bigtriangledown(A1 \times b),\ \triangle(A2 \times b)], & \text{otherwise.} \end{cases}$
Exceptions: All caused by exceptional operands and exceptional results on floating-point multiplications. Handling by the corresponding floating-point system according to the IEEE Std. In addition, *Invalid Operation* exception may be signaled if the real operand is a NaN. The delivered default result is [qNaN, qNaN] if the exception occurs without a trap.

GiasMulII(INTERVAL R, INTERVAL A, INTERVAL B)
Purpose: Calculates the directed interval R including the true result of $A \times B$.
Description:

$$R = \begin{cases} \text{If GiasSign(A)} \neq 0 \text{ and GiasSign(B)} \neq 0 \text{ then} \\ [\bigtriangledown(A1 \times B1),\ \triangle(A2 \times B2)], \text{ if GiasSign(A)=GiasSign(B)=1} \\ [\bigtriangledown(A2 \times B1),\ \triangle(A1 \times B2)], \text{ if GiasSign(A)=1, GiasSign(B)=}-1 \\ [\bigtriangledown(A1 \times B2),\ \triangle(A2 \times B1)], \text{ if GiasSign(A)=}-1\text{, GiasSign(B)=1} \\ [\bigtriangledown(A2 \times B2),\ \triangle(A1 \times B1)], \text{ if GiasSign(A)=GiasSign(B)=}-1 \\ \text{If GiasSign(A)} \neq 0 \text{ and GiasSign(B)=0 then} \\ [\bigtriangledown(A2 \times B1),\ \triangle(A2 \times B2)], \text{ if GiasSign(A)=1, GiasDirection(B)=1} \\ [\bigtriangledown(A1 \times B2),\ \triangle(A1 \times B1)], \text{ if GiasSign(A)=}-1\text{, GiasDirection(B)=1} \\ [\bigtriangledown(A1 \times B1),\ \triangle(A1 \times B2)], \text{ if GiasSign(A)=1, GiasDirection(B)=}-1 \\ [\bigtriangledown(A2 \times B2),\ \triangle(A2 \times B1)], \text{ if GiasSign(A)=}-1\text{, GiasDirection(B)=}-1 \\ [\bigtriangledown(A1 \times B1),\ \triangle(A2 \times B2)], \text{ if GiasDirection(B)=0} \\ \text{If GiasSign(A)=0 and GiasSign(B)} \neq 0 \text{ then} \\ [\bigtriangledown(A1 \times B2),\ \triangle(A2 \times B2)], \text{ if GiasDirection(A)=1, GiasSign(B)=1} \\ [\bigtriangledown(A2 \times B1),\ \triangle(A1 \times B1)], \text{ if GiasDirection(A)=1, GiasSign(B)=}-1 \\ [\bigtriangledown(A1 \times B1),\ \triangle(A2 \times B1)], \text{ if GiasDirection(A)=}-1\text{, GiasSign(B)=1} \\ [\bigtriangledown(A2 \times B2),\ \triangle(A1 \times B2)], \text{ if GiasDirection(A)=}-1\text{, GiasSign(B)=}-1 \\ [\bigtriangledown(A1 \times B1),\ \triangle(A2 \times B2)], \text{ if GiasDirection(A)=0} \\ \text{If GiasSign(A)=GiasSign(B)=0 then} \\ [\min\{\bigtriangledown(A1 \times B2), \bigtriangledown(A2 \times B1)\},\ \max\{\triangle(A1 \times B1), \triangle(A2 \times B2)\}], \\ \qquad \text{if GiasDirection(A)=GiasDirection(B)=1} \\ [\max\{\bigtriangledown(A1 \times B1), \bigtriangledown(A2 \times B2)\},\ \min\{\triangle(A1 \times B2), \triangle(A2 \times B1)\}], \\ \qquad \text{if GiasDirection(A)=GiasDirection(B)=}-1 \\ [0,\ 0], \qquad \text{if GiasDirection(A)} \times \text{GiasDirection(B)=}-1 \\ [\bigtriangledown(A1 \times B1),\ \triangle(A2 \times B2)], \qquad \text{if GiasDirection(A)} \times \text{GiasDirection(B)=0} \end{cases}$$

Exceptions: Al caused by exceptional operands and exceptional results on floating-point multiplications. Handling by the corresponding floating-point system according to the IEEE Std. In addition, *Invalid Operation* exception may be signaled if some of the operands involve NaN. The default result should involve at least one qNaN as end-point.

GiasDivRR(INTERVAL R, REAL a, REAL b)
Purpose: Calculates the directed interval R including the true result of a/b.
Description: $R = [\bigtriangledown(a/b),\ \triangle(a/b)]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point divisions. Handling by the corresponding floating-point system according to the IEEE Std.

GiasDivRI(INTERVAL R, REAL a INTERVAL B)
Purpose: Calculates the directed interval R including the true result of a/B.
Description: R = GasDivII(A, B), where $A = [a,\ a]$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point divisions. Handling by the corresponding floating-point system according to the IEEE Std. In addition, *Invalid Operation* exception may be signaled if some of the operands involve NaN. The default result should involve at least one qNaN as end-point. *Division by Zero* exception is signaled if ± 0 is in the interior of the proper part of B. The delivered default result is [qNaN, qNaN] if the exception occurs without a trap .

GiasDivIR(INTERVAL R, INTERVAL A, REAL b)
Purpose: Calculates the directed interval R including the true result of A/b.
Description: $R = \begin{cases} [\bigtriangledown(A2/b),\ \triangle(A1/b)], & \text{if } b < 0 \text{ or } b = -0 \\ [\bigtriangledown(A1/b),\ \triangle(A2/b)], & \text{otherwise.} \end{cases}$

Exceptions: All caused by exceptional operands and exceptional results on floating-point divisions. Handling by the corresponding floating-point system according to the IEEE Std. In addition, *Invalid Operation* exception may be signaled if the real operand is a NaN. The delivered default result is [qNaN, qNaN] if the exception occurs without a trap.

GiasDivII(INTERVAL R, INTERVAL A, INTERVAL B)

Purpose: Calculates the directed interval R including the true result of A/B.

Description:

$$R = \begin{cases} \text{If GiasSign(A)}\neq 0 \ \text{ and } \ \text{GiasSign(B)}\neq 0 \ \text{ then} & \\ [\bigtriangledown(A1/B2),\ \triangle(A2/B1)], & \text{if GiasSign(A)=GiasSign(B)=1} \\ [\bigtriangledown(A2/B2),\ \triangle(A1/B1)], & \text{if GiasSign(A)=1, GiasSign(B)=}-1 \\ [\bigtriangledown(A1/B1),\ \triangle(A2/B2)], & \text{if GiasSign(A)=}-1\text{, GiasSign(B)=1} \\ [\bigtriangledown(A2/B1),\ \triangle(A1/B2)], & \text{if GiasSign(A)=}-1\text{, GiasSign(B)=}-1 \\ \text{If GiasSign(A)=0 and GiasSign(B)}\neq 0 \ \text{ then} & \\ [\bigtriangledown(A1/B1),\ \triangle(A2/B1)], & \text{if GiasDirection(A)=1, GiasSign(B)=1} \\ [\bigtriangledown(A2/B2),\ \triangle(A1/B2)], & \text{if GiasDirection(A)=1, GiasSign(B)=}-1 \\ [\bigtriangledown(A1/B2),\ \triangle(A2/B2)], & \text{if GiasDirection(A)=}-1\text{, GiasSign(B)=1} \\ [\bigtriangledown(A2/B1),\ \triangle(A1/B1)], & \text{if GiasDirection(A)=}-1\text{, GiasSign(B)=}-1 \\ [\bigtriangledown(A1/B1),\ \triangle(A2/B2)], & \text{if GiasDirection(A)=0} \\ \text{If GiasSign(B)=0 then} & \\ [\bigtriangledown(A1/B1),\ \triangle(A2/B2)], & \text{if GiasDirection(B)=0} \\ [\texttt{qNaN},\ \texttt{qNaN}], & \text{if GiasDirection(B)} \neq 0 \end{cases}$$

Exceptions: All caused by exceptional operands and exceptional results on floating-point divisions. Handling by the corresponding floating-point system according to the IEEE Std. In addition, *Invalid Operation* exception may be signaled when some of the operands involves NaN. The default result should involve at least one qNaN as end-point. *Division by Zero* exception is signaled if ± 0 is in the interior of the proper part of B. The delivered default result is [qNaN, qNaN] if the exception occurs without a trap.

GiasDual(INTERVAL R, INTERVAL A)

Purpose: Conjugation of the directed interval A.

Description: $R = [A2,\ A1]$.

Exceptions: Depend on how the implementation considers copying operation. When copying operation is implemented as an arithmetic operation then all the exceptions are caused by the corresponding floating-point exceptional operands. There will be no exceptions if copying is treated as non-arithmetic operation.

INTEGER GiasDirection(INTERVAL A)

Purpose: Returns -1, if A is an improper directed interval; 1, if A is proper or degenerate and 0, if A involves NaN.

Description: If $A1 \leq A2$ then If ($A1 = 0$ and $A2 = -0$) then -1 else 1
elseIf ($A1$ =NaN or $A2$ =NaN) then 0 else -1.

Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in the argument. 0 is delivered as a result if the exception occurs without a trap.

INTEGER GiasSign(INTERVAL A)

Purpose: Returns -1, if A is a negative directed interval; 1, if A is positive and 0, if 0 is in the interior of the proper part of A or A involves NaN.

Description: If $A1 \leq 0$ and $A2 \leq 0$ then

$$\text{If } \begin{pmatrix} A1 = 0 \text{ and } A2 = 0 \text{ or} \\ A1 = 0 \text{ and } (A2 = -0 \text{ or } A2 < 0) \text{ or} \\ (A1 < 0 \text{ or } A1 = -0) \text{ and } A2 = 0 \end{pmatrix} \text{ then } 0 \text{ else } -1$$

elseIf $A1 \geq 0$ and $A2 \geq 0$ then

$$\text{If } \begin{pmatrix} A1 = -0 \text{ and } (A2 = 0 \text{ or } A2 > 0) \text{ or} \\ (A1 > 0 \text{ or } A1 = 0) \text{ and } A2 = -0 \end{pmatrix} \text{ then } 0 \text{ else } 1$$

else 0.

Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in the argument. 0 is delivered as a result if the exception occurs without a trap.

GiasProper(INTERVAL R, INTERVAL A)

Purpose: Returns the proper part (projection) of the directed interval A.

Description: $R = \begin{cases} \text{GiasDual(R, A)}, & \text{if GiasDirection(A) ¡ 0}, \\ \text{A}, & \text{otherwise.} \end{cases}$

Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in the argument. The argument is returned unchanged if the exception occurs without a trap.

REAL GiasInf(INTERVAL A)

Purpose: $\text{Inf}(A) := \min\{A1, A2\}$ returns the lower bound of the directed interval A.

Description: $\begin{cases} A1, & \text{if } A1 \leq A2 \\ A2, & \text{if } A1 > A2 \\ \text{qNaN}, & \text{otherwise.} \end{cases}$

Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in the argument. qNaN is delivered as a result if the exception occurs without a trap.

REAL GiasSup(INTERVAL A)

Purpose: $\text{Sup}(A) := \max\{A1, A2\}$ returns the upper bound of the directed interval A.

Description: $\begin{cases} A2, & \text{if } A1 \leq A2 \\ A1, & \text{if } A1 > A2 \\ \text{qNaN}, & \text{otherwise.} \end{cases}$

Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in the argument. qNaN is delivered as a result if the exception occurs without a trap.

REAL GiasFirst(INTERVAL A)

Purpose: Returns the first end-point of the directed interval A.

Description: $A1$.

Exceptions: Depend on whether copying is implemented as an arithmetic operation or not.

REAL GiasSecond(INTERVAL A)

Purpose: Returns the second end-point of the directed interval A.

Description: $A2$.

Exceptions: Depend on whether copying is implemented as an arithmetic operation or not.

REAL GiasMid(INTERVAL A)

Purpose: Returns the nearest value to the midpoint of a directed interval A.

Description: $(A1 + A2)/2$.

Exceptions: All caused by exceptional operands and exceptional results on floating-point operations. Handling by the corresponding floating-point system according to the IEEE Std.

REAL GiasDiam(INTERVAL A)
Purpose: Returns the nearest value to the diameter of a directed interval A.
$\mathrm{Diam}(A) := \mid A1 - A2 \mid$.
Description: $\mathrm{Abs}(A1 - A2)$.
Exceptions: All caused by exceptional operands and exceptional results on floating-point subtraction. Handling by the corresponding floating-point system according to the IEEE Std.

REAL GiasAbs(INTERVAL A)
Purpose: Returns the absolute value of the directed interval A.
$\mathrm{Abs}(A) := \max\{\mid A1 \mid, \mid A2 \mid\}$.
Description: $\begin{cases} \mathrm{Abs}(A1), & \text{if } \mathrm{Abs}(A1) \geq \mathrm{Abs}(A2), \\ \mathrm{Abs}(A2), & \text{if } \mathrm{Abs}(A1) < \mathrm{Abs}(A2), \\ \text{qNaN}, & \text{otherwise.} \end{cases}$
Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in the argument. qNaN is delivered if the exception arises without a trap.

REAL GiasDistRI(REAL a, INTERVAL B)
Purpose: Returns the distance between a and B.
$\mathrm{Dist}(a, B) := \max\{\mid a - B1 \mid, \mid a - B2 \mid\}$.
Description: $\begin{cases} \mathrm{Abs}(a - B1), & \text{if } \mathrm{Abs}(a - B1) \geq \mathrm{Abs}(a - B2), \\ \mathrm{Abs}(a - B2), & \text{if } \mathrm{Abs}(a - B1) < \mathrm{Abs}(a - B2), \\ \text{qNaN}, & \text{otherwise.} \end{cases}$
Exceptions: All caused by exceptional operands and exceptional results on floating-point subtractions. Handling by the corresponding floating-point system according to the IEEE Std. In addition, *Invalid Operation* exception may be signaled if NaN is involved in some of the arguments. qNaN is delivered as a default result if the exception occurs without a trap.

REAL GiasDistII(INTERVAL A, INTERVAL B)
Purpose: Returns the distance between A and B.
$\mathrm{Dist}(A, B) := \max\{\mid A1 - B1 \mid, \mid A2 - B2 \mid\}$.
Description: $\begin{cases} \mathrm{Abs}(A1 - B1), & \text{if } \mathrm{Abs}(A1 - B1) \geq \mathrm{Abs}(A2 - B2), \\ \mathrm{Abs}(A2 - B2), & \text{if } \mathrm{Abs}(A1 - B1) < \mathrm{Abs}(A2 - B2), \\ \text{qNaN}, & \text{otherwise.} \end{cases}$
Exceptions: All caused by exceptional operands and exceptional results on floating-point subtractions. Handling by the corresponding floating-point system according to the IEEE Std. In addition, *Invalid Operation* exception may be signaled if NaN is involved in some of the arguments. qNaN is delivered as a default result if the exception occurs without a trap.

GiasIntersection(INTERVAL R, INTERVAL A, INTERVAL B)
Purpose: Calculates the intersection R between directed interval A and directed interval B.
$R := [\max\{A1, B1\}, \min\{A2, B2\}]$.
Description: If A1≥B1 then If A1= −0 and B1= 0 then R1=B1 else R1=A1
elseIf A1=NaN or B1=NaN then R1=qNaN else R1=B1
If A2≤B2 then If A2= 0 and B2= −0 then R2=B2 else R2=A2
elseIf A2=NaN or B2=NaN then R2=qNaN else R2=B2.
Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in some of the arguments. If the exception occurs without a trap, the delivered default result involves qNaN at that end-point at which NaN is involved in the arguments.

GiasHullR(INTERVAL R, REAL a)
Purpose: Composes interval R with a as end-points.

Description: $R = [a, a]$.
Exceptions: Depend on whether copying is implemented as an arithmetic operation or not.

GiasHullRR(INTERVAL R, REAL a, REAL b)
Purpose: Initializes R with the convex hull of a and b. $R := [\min\{a, b\}, \max\{a, b\}]$ is always a proper interval.
Description: If $a \geq b$ then If ($a = -0$ and $b = 0$) then $R = [b, a]$ else $R = [a, b]$ elseIf (a=NaN or b=NaN) then R =[qNaN, qNaN] else $R = [b, a]$.
Exceptions: *Invalid Operation* exception may be signaled if some of the arguments is NaN. [qNaN, qNaN] is delivered if the exception occurs without a trap.

GiasHullRI(INTERVAL R, REAL a, INTERVAL B)
Purpose: Initializes R with the convex hull of a and B. $R := [\min\{a, B1\}, \max\{a, B2\}]$ is always a proper interval.
Description: R=GiasHullII(A, B), where $A = [a, a]$.
Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in some of the arguments. If the exception occurs without a trap, the delivered default result involves qNaN at that end-point at which NaN is involved in the arguments.

GiasHullIR(INTERVAL R, INTERVAL A, REAL b)
Purpose: Initializes R with the convex hull of A and b. $R := [\min\{A1, b\}, \max\{A2, b\}]$ is always a proper interval.
Description: R=GiasHullII(A, B), where $B = [b, b]$.
Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in some of the arguments. If the exception occurs without a trap, the delivered default result involves qNaN at that end-point at which NaN is involved in the arguments.

GiasHullII(INTERVAL R, INTERVAL A, INTERVAL B)
Purpose: Initializes R with the convex hull of A and B. $R := [\min\{A1, B1\}, \max\{A2, B2\}]$ is always a proper interval.
Description: If A1≤B1 then If (A1= 0 and B1= −0) then R1=B1 else R1=A1
elseIf (A1=NaN or B1=NaN) then R1=qNaN else R1=B1
If A2≥B2 then If (A2= −0 and B2= 0) then R2=B2 else R2=A2
elseIf (A2=NaN or B2=NaN) then R2=qNaN else R2=B2.
Exceptions: *Invalid Operation* exception may be signaled if NaN is involved in some of the arguments. If the exception occurs without a trap, the delivered default result involves qNaN at that end-point at which NaN is involved in the arguments.

INTEGER GiasEqualRI(REAL a, INTERVAL B)
Purpose: Tests equality of a real value and a directed interval.
Description: $\begin{cases} 1, & \text{if } a = B1 \text{ and } a = B2, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: None.

INTEGER GiasEqualIR(INTERVAL A, REAL b)
Purpose: Tests equality of a directed interval and a real value.
Description: $\begin{cases} 1, & \text{if } A1 = b \text{ and } A2 = b, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: None.

INTEGER GiasEqualII(INTERVAL A, INTERVAL B)
Purpose: Tests equality of two directed intervals.
Description: $\begin{cases} 1, & \text{if } A1 = B1 \text{ and } A2 = B2, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: None.

INTEGER GiasInRI(REAL a, INTERVAL B)
Purpose: Tests a real value to be included in a directed interval.
$(a \subseteq B \ :\Longleftrightarrow\ B1 \leq a \text{ and } a \leq B2)$.
Description: $\begin{cases} 1, & \text{if } B1 \leq a \text{ and } a \leq B2, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasInIR(INTERVAL A, REAL b)
Purpose: Tests a directed interval to be included in a real value.
$(A \subseteq b \ :\Longleftrightarrow\ b \leq A1 \text{ and } A2 \leq b)$.
Description: $\begin{cases} 1, & \text{if } b \leq A1 \text{ and } A2 \leq b, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasInII(INTERVAL A, INTERVAL B)
Purpose: Tests $A \subseteq B$. $(A \subseteq B \ :\Longleftrightarrow\ B1 \leq A1 \text{ and } A2 \leq B2)$.
Description: $\begin{cases} 1, & \text{if } B1 \leq A1 \text{ and } A2 \leq B2, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasInteriorRI(REAL a, INTERVAL B)
Purpose: Tests the antireflexive order relation "contained in".
$a \subset B \ :\Longleftrightarrow\ a \subseteq B \text{ and } a \neq B$.
Description: $\begin{cases} 1, & \text{if GiasInRI(a, B)} = 1 \ \text{ and } \ \text{GiasEqualRI(a, B)} = 0, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasInteriorIR(INTERVAL A, REAL b)
Purpose: Tests the antireflexive order relation "contained in".
$A \subset b \ :\Longleftrightarrow\ A \subseteq b \text{ and } A \neq b$.
Description: $\begin{cases} 1, & \text{if GiasInIR(A, b)} = 1 \ \text{ and } \ \text{GiasEqualIR(A, b)} = 0, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasInteriorII(INTERVAL A, INTERVAL B)
Purpose: Tests the antireflexive order relation "contained in".
$A \subset B \ :\Longleftrightarrow\ A \subseteq B \text{ and } A \neq B$.
Description: $\begin{cases} 1, & \text{if GiasInII(A, B)} = 1 \ \text{ and } \ \text{GiasEqualII(A, B)} = 0, \\ 0, & \text{otherwise.} \end{cases}$

Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasLessEqRI(REAL a, INTERVAL B)
Purpose: Tests $a \le B$ ($a \le B \;:\Longleftrightarrow\; a \le B1$ and $a \le B2$).
Description: $\begin{cases} 1, & \text{if } (a \le B1) \text{ and } (a \le B2) \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasLessEqIR(INTERVAL A, REAL b)
Purpose: Tests $A \le b$ ($A \le b \;:\Longleftrightarrow\; A1 \le b$ and $A2 \le b$).
Description: $\begin{cases} 1, & \text{if } (A1 \le b) \text{ and } (A2 \le b) \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasLessEqII(INTERVAL A, INTERVAL B)
Purpose: Tests $A \le B$ ($A \le B \;:\Longleftrightarrow\; A1 \le B1$ and $A2 \le B2$).
Description: $\begin{cases} 1, & \text{if } (A1 \le B1) \text{ and } (A2 \le B2) \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasLessRI(REAL a, INTERVAL B)
Purpose: Tests the antireflexive order relation $a < B$.
($a < B \;:\Longleftrightarrow\; a \le B$ and $a \ne B$).
Description: $\begin{cases} 1, & \text{if GiasLessEqRI(a, B)} = 1 \text{ and GiasEqualRI(a, B)} = 0, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasLessIR(INTERVAL A, REAL b)
Purpose: Tests the antireflexive order relation $A < b$.
($A < b \;:\Longleftrightarrow\; A \le b$ and $A \ne b$).
Description: $\begin{cases} 1, & \text{if GiasLessEqIR(A, b)} = 1 \text{ and GiasEqualIR(A, b)} = 0, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

INTEGER GiasLessII(INTERVAL A, INTERVAL B)
Purpose: Tests the antireflexive order relation $A < B$.
($A < B \;:\Longleftrightarrow\; A \le B$ and $A \ne B$).
Description: $\begin{cases} 1, & \text{if GiasLessEqII(A, B)} = 1 \text{ and GiasEqualII(A, B)} = 0, \\ 0, & \text{otherwise.} \end{cases}$
Exceptions: Those on floating-point comparison operations, i.e. *Invalid Operation* if NaN is involved in some of the operands. Zero is delivered as a result if the exception occurs without a trap.

Acknowledgements

This work was partially supported by the Swiss National Science Foundation and the Bulgarian National Science Fund under grant No. I-507/95.

References

[Adams and Kulisch 1993] Adams, E.; Kulisch, U. (Eds.): "Scientific Computing with Automatic Result Verification. I. Language and Programming Support for Verified Scientific Computation, II. Enclosure Methods and Algorithms with Automatic Result Verification, III. Applications in the Engineering Sciences"; Academic Press, San Diego (1993).

[Alefeld and Herzberger 1974] Alefeld, G.; Herzberger, J.: "Einführung in die Intervallrechnung"; Bibliographisches Institut Mannheim (1974).

[ANSI/IEEE 1985] American National Standards Institute/Institute of Electrical and Electronics Engineers: "IEEE Standard for Binary Floating-Point Arithmetic"; ANSI/IEEE Std 754–1985, New York (1985).

[ANSI/IEEE 1987] American National Standards Institute/Institute of Electrical and Electronics Engineers: "IEEE Standard for Radix-Independent Floating-Point Arithmetic"; ANSI/IEEE Std 854–1987, New York (1987).

[Corliss 1990] Corliss, G. F.: "Proposal for a Basic Interval Arithmetic Subroutines Library (BIAS)"; Technical Report, Marquette University Department of Mathematics, Statistics and Computer Science, Milwaukee, Wisc. (1991).

[Dimitrova et al. 1992] Dimitrova, N.; Markov, S. M.; Popova, E.: "Extended Interval Arithmetics: New Results and Applications"; In Atanassova, L.; Herzberger, J. (Eds.): "Computer Arithmetic and Enclosure Methods"; Elsevier Sci. Publishers B. V. (1992), 225-232.

[Dongara et al. 1985] Dongara J. J.; Croz, J. D.; Hammarling, S. and Hanson, R. J.: "A Proposal for an Extended Set of FORTRAN Basic Linear Algebra Subprograms"; ACM SIGNUM Newsletter, Vol. 20, No. 1 (1985), 2-18.

[Durst 1975] Durst, E.: "Realisierung einer erweiterten Intervallrechnung mit Überlaufarithmetik"; Diplomarbeit, Universität Karlsruhe (1975).

[Gardenes and Trepat 1980] Gardeñes, E.; Trepat, A.: "Fundamentals of SIGLA, an Interval Computing System over the Completed Set of Intervals"; Computing, 24 (1980), 161-179.

[Goguen et al. 1978] Goguen, J. A.; Thatcher, J. W. and Wagner, E. G.: "An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types"; In: Raymond T. Yeh (Ed.): "Current Trends in Programming Methodology IV"; Prentice-Hall (1978).

[Kaucher 1973] Kaucher, E.: "Über metrische und algebraische Eigenschaften einiger beim numerischen Rechnen auftretenden Räume"; Dissertation, Universität Karlsruhe (1973).

[Kaucher 1977] Kaucher, E.: "Über Eigenschaften und Anwendungsmöglichkeiten der erweiterten Intervallrechnung und des hyperbolischen Fastkörpers über R"; Computing Suppl., 1 (1977), 81-94.

[Kaucher 1980] Kaucher, E.: "Interval Analysis in the Extended Interval Space IR"; Computing Suppl., 2 (1980), 33-49.

[Kearfott et al. 1996] Kearfott, R. Baker et al.: "A Specific Proposal for Interval Arithmetic in Fortran"; March 20, 1996, electronic submission to the "Reliable Computing" newsgroup.

[Klatte et al. 1992] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: "PASCAL-XSC Language Reference with Examples"; Springer, Berlin (1992).

[Knueppel 1993] Knüppel, O.: "BIAS — Basic Interval Arithmetic Subroutines"; Bericht 93.3, Technische Universität Hamburg-Harburg, Hamburg (1993).

[Knueppel 1994] Knüppel, O.: "PROFIL/BIAS — A Fast Interval Library"; Computing, 53 (1994), 277-287.

[Kulisch and Miranker 1981] Kulisch, U., Miranker, W. L.: "Computer Arithmetic in Theory and Practice"; Academic Press, New York (1981).

[Kupriyanova 1995] Kupriyanova, L.: "Inner Estimation of the United Solution Set of Interval Linear Algebraic System"; Reliable Computing, 1, No. 1 (1995), 15-31.

[Lawson et al. 1979] Lawson, C. L.; Hanson, R. J.; Kincaid, D. R.; Krogh, F. T.: "Basic Linear Algebra Subprograms for Fortran Usage", ACM Transaction on Mathematical Software, Vol. 5, No. 3 (1979), 308-323.

[Markov 1992] Markov, S. M.: "On the Presentation of Ranges of Monotone Functions Using Interval Arithmetic"; Interval Computations, No. 4 (1992), 19-31.

[Markov 1993] Markov, S. M.: "Some Interpolation Problems Involving Interval Data"; Interval Computations, No. 3 (1993), 164-182.

[Markov 1995] Markov, S. M.: "On Directed Interval Arithmetic and its Applications"; J. UCS, 1, 7 (1995), 510-521.

[Markov et al. 1995] Markov, S., E. Popova, Ch. Ullrich: "On the Solution of Linear Algebraic Equations Involving Interval Coefficients"; In: Margenov, S. and Vassilevski, P. (Eds.): "Iterative Methods in Linear Algebra, II"; IMACS Series in Computational and Applied Mathematics, Vol. 3 (1996), 216-225.

[Moore 1966] Moore, R. E.: "Interval Analysis"; Prentice-Hall, Englewood Cliffs, N. J. (1966).

[Nesterov 1995] Nesterov , V.: "Implementation of Generalized Interval Arithmetics"; Abstracts IMACS/GAMM International Symposium SCAN-95, September 26-29, Wuppertal, Germany (1995), 97.

[Parnas 1972] Parnas, D. L.: "A Technique for software Module Specification with Examples"; Communications of the ACM, Vol. 15, No. 5 (1972), 330-336.

[Popova 1994] Popova, E.: "Extended Interval Arithmetic in IEEE Floating-Point Environment"; Interval Computations, No. 4 (1994), 100-129.

[Popova 1996] Popova, E.: "Interval Operations Involving NaNs"; Reliable Computing, 2, No. 2 (1996), 161-166.

[Shary 1996a] Shary, S. P.: "Algebraic Approach to the Interval Linear Static Identification, Tolerance and Control Problems, or One More Application of Kaucher Arithmetic"; Reliable Computing, 2, No. 1 (1996), 1-32.

[Shary 1996b] Shary, S. P.: "A New Approach to the Analysis of Interval Static Systems Under Interval Uncertainty"; In: Alefeld, G.; Frommer, A.; Lang, B. (Eds.): "Scientific Computing and Validated Numerics"; Akademie-Verlag, Berlin, (1996), 118-132.

[Ullrich 1994] Ullrich, C. P.: "Interval Arithmetic on Computers"; In: Herzberger, J. (Ed.): "Topics in Validated Computations"; Elsevier Science B. V. (1994), 473-497.