Latest Developments on the IEEE 1788 Effort for the Standardization of Interval Arithmetic

Nathalie Revol

Latest Developments on the IEEE 1788 Effort for the Standardization of Interval Arithmetic

N. Revol
[1]INRIA, LIP (UMR 5668 CNRS - ENS de Lyon - INRIA - UCBL), University of Lyon
LIP, École Normale Supérieure de Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France
email: Nathalie.Revol@ens-lyon.fr
and the IEEE P1788 Working Group
http://grouper.ieee.org/groups/1788/

December 15, 2013

Abstract

Interval arithmetic undergoes a standardization effort started in 2008 by the IEEE P1788 working group. The structure of the proposed standard is presented: the mathematical level is distinguished from both the implementation and representation levels. The main definitions are introduced: what is an interval? how are mathematical functions, such as arithmetic operations or trigonometric functions, extended over intervals? what are the comparison relations? how are set operations, such as the union or the intersection, extended over intervals?

While developing this standard, some topics led to hot debates: these topics correspond to points most delicate and difficult to define. Such a hot topic is the handling of exceptions. Eventually, the system of decorations has been adopted. A decoration is a piece of information that is attached to each interval. Rules for the propagation of decorations have also been defined by the standardization group. Another hot topic is the mathematical model used for interval arithmetic. Historically, the model introduced by R. Moore in the 60s covered only non-empty and bounded intervals. The set-based model includes the empty set and unbounded intervals as well. Tenants of Kaucher arithmetic also insisted on offering "reverse" intervals. It has eventually been decided that an implementation must provide at least one of these flavors of interval arithmetic. The standard provides hooks for these different flavors.

Finally, a tentative list of missing items is given. As the preparation of the draft should end in December 2013, no chapter is missing. However, a reference implementation would be welcome to validate the choices made during the development of the standard for interval arithmetic.

1 Standardization effort

Interval arithmetic is a tool of choice to implement set computing as well as guaranteed numerical computing. It offers two unique features indeed: the inclusion property and an effective version of the theorem of Schauder.

The inclusion property is also called the fundamental theorem of interval arithmetic. It states that, for any operation, the result of this operation applied to interval operands contains the set of results obtained by picking one number as operand in each interval operand. This theorem is sometimes also referred to as the "Thou Shalt not Lie" principle. Examples of particular importance are interval operands which correspond to data with uncertainty (such as measurement uncertainty) or to numbers that are not representable in the given format and are represented as enclosing intervals. Large sets of values also fit in this framework: interval arithmetic makes it possible to compute with sets.

Another extremely valuable feature of interval arithmetic is that it provides an effective version of the theorem of Schauder. This theorem states that every continuous function from a convex compact subset of a

Banach space into itself has a fixed point. Interval arithmetic offers a means to check whether the conditions of application of this theorem hold. A non-empty bounded interval I of $\mathbb{R}^n$ is a convex compact subset of $\mathbb{R}^n$. If interval computations for $f(I)$ with a continuous function f yield an interval that is a subset of I, it is then guaranteed that $f(I) \subset I$. The theorem of Schauder applies and this constitutes a proof that f has a fixed point in I. This is very useful in Newton-like methods, where the very existence of a zero is proven meanwhile an interval enclosure is obtained.

These two features make interval arithmetic a highly valuable tool. Unfortunately, libraries and software based on interval arithmetic do not (yet) share enough definitions and operations, and do not (yet) offer enough functionalities to be widely used. However, at a seminar in Dagstuhl, Germany, in January 2008, the audience felt confident that the field of interval arithmetic was mature enough and that individual points of views had converged (Kearfott, Pryce, Revol 2009). In other words, it was felt that interval arithmetic was ready to undergo a standardization effort. The working group for the standardization of interval arithmetic was launched (IEEE P1788) in 2008 under the auspices of IEEE, under the name IEEE P1788 ("P" stands for " Project", indicating that the corresponding standard is not yet finalized) WG ("WG" for "Working Group"). IEEE was chosen because it is the institute where the standard for floating-point arithmetic was developed: the first version is IEEE-754, established in 1985 (IEEE 754-1985), and a substantial revision and enlargement was concluded in 2008 (IEEE 754-2008). For the standardization of interval arithmetic, the deadline granted by IEEE is December 2014. A number of issues, correctly predicted by (Edmonson, Melquiond 2009), have been addressed. From 2008 to 2013, numerous discussions, conducted electronically through a mailing list with above 140 subscribers, led to the proposal of over 50 motions and to the elaboration of a draft text for the standard. The structure and the main lines of this standard are sketched in the next two sections. The hottest topics of discussion are summed up afterwards. The draft text is close to completion as of December 2013. Most of the year 2014 will be devoted to wordsmithing the text of the standard and to submitting it to an internal ballot, then to the scrutiny of a group of experts appointed by IEEE.

2 Structuration into levels

It was decided to clearly distinguish 4 levels for the specification of interval arithmetic. The structuration is copied from the IEEE 754 standard for floating-point arithmetic (IEEE 754-2008). Level 1 corresponds to the mathematical theory of interval arithmetic: definition of an interval, of an operation.... This level also defines decorations, at the heart of the mechanism for exceptions, which are detailed below. Level 2 defines the implementation of the mathematical theory. It specifies how the entities defined at Level 1 can be mapped onto a finite set of machine representable data. Level 2 also states what additional information must be known, such as the type of the endpoints if an interval is represented by its endpoints. In this case, the numeric type used for the endpoints must contain 0, infinities, be symmetric around 0 and satisfy several other requirements specified at Level 2. Level 2 also defines the behavior of the operations which are mathematicall defined at Level 1, when they operate on objects of Level 2. For instance, the operation that returns the midpoint of an interval has no value at Level 1 when applied to the empty interval, it must have one at Level 2: it has been decided that $\mathrm{mid}(\emptyset) = \mathrm{NaN}$ (Not a Number) at Level 2. Level 3 concerns only representation issues, such as conversions between formats or input/output. It also specifies how representations influence the implementations of operations. For instance, if floating-point numbers are used to represent intervals, say by their endpoints, then the sum $[\underline{a}, \bar{a}]+[\underline{b}, \bar{b}]$ is represented by $[\mathrm{RD}(\underline{a}+\underline{b}), \mathrm{RU}(\bar{a}+\bar{b})]$ where RD stands for rounding downwards and RU for rounding upwards: implementation issues, such as roundings, in this example, are accounted for at Level 3. Level 4 is about encoding, at the bit level. In the IEEE-754 standard for floating-point arithmetic, Level 4 is fully specified. However, it appeared that a standard for interval arithmetic relies on floating-point numbers or on other representations for numbers: as these representations define the bit-level representations for numbers, a standard for interval arithmetic does not have to rule anything at Level 4.

Fig. 1 sketches the structure into 4 levels and the content of each level. Next section contains a more detailed presentation of these contents.

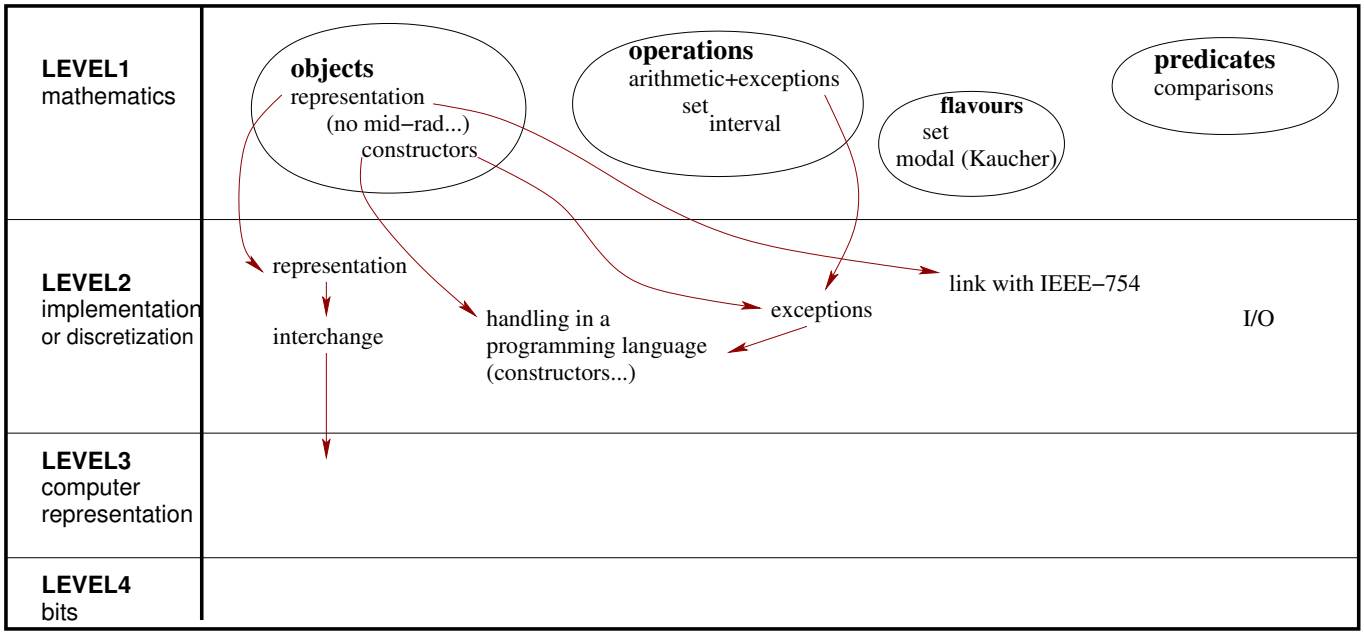

Figure 1: Structuration of the standard for interval arithmetic in 4 levels.

3 What is specified by the standard

Let us denote an interval in boldface: x is a real, $\mathbf{x}$ is an interval.

The standard for interval arithmetic is designed to integrate smoothly different mathematical theories: each such available theory is called a *flavor*. A minimal common basis has been defined: cases have been identified where, for all flavors, computations must return the same result. Unsurprisingly, these cases cover operations applied to bounded intervals included in their definition domains.

The standard first specifies what an interval is. It was decided to extend Moore's definition by defining, at Level 1, an interval as a closed, connected subset of $\mathbb{R}$. This includes the empty set, bounded intervals $[\underline{a}, \bar{a}]$ with $\underline{a} \in \mathbb{R}$, $\bar{a} \in \mathbb{R}$ and $\underline{a} \leq \bar{a}$, and unbounded intervals. Constructors accept as inputs either a pair (l, u) of real numbers and construct the interval $[l, u]$, and l and u must verify $l \leq u$ in the set-based flavor, or a string that can be read as an interval. Exceptional cases are handled properly.

Then some mathematical operations and functions are specified by the standard while other functions are only recommended. The meaning of the evaluation of one of these operations or functions, at Level 1, is the following: if $\mathbf{x}$ is an interval or an interval vector and f an operation or function that is applied to $\mathbf{x}$, $f(\mathbf{x})$ is to be understood as $f(\mathbf{x} \cap \mathrm{Dom}_f)$ where Dom_f denotes the definition domain of f. Examples are $[1,2] + [3,5] = [4,7]$, $\sqrt{[-2,1]} = [0,1]$ and $1/[0,0] = \emptyset$. Required arithmetic functions are $+$, $-$, $\times$, $/$, $1/$, ^{2}, $\sqrt{}$ and fma. Required mathematical functions are pow (with some restrictions on the domain), exp, $\exp_1$, $\exp_{10}$, log, $\log_2$, $\log_{10}$, sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, asinh, acosh, atanh, sign, ceil, floor, trunc, roundTiesToEven and roundTiesToAway, abs, min, max. Another division, that returns 0, 1 or 2 intervals, is also required. This division is especially useful for Newton's algorithm, where it is relevant to know when a division by 0 occurred. In this case, the result may have a gap, and preserving this gap is useful to isolate roots before proceeding with the following steps. For the set-based flavor, the standard thus specifies `divToPair` which offers this functionality. The definition corresponds to the usual definition in set theory. Recommended functions in the set-based flavor, but not mandatory, are rootn, expm1, exp2m1, exp10m1, logp1, log2p1, log10p1, compoundm1, hypot, rSqrt, sinPi, cosPi, tanPi, asinPi, acosPi, atanPi, atan2Pi. These lists correspond more or less to the floating-point operations and functions defined in IEEE-

754 for floating-point arguments (either required or recommended). Furthermore, slope functions are also recommended, in the set-based flavor, for exp, log, cos, sin, asin, atan, cosh and sinh, with order 1 or 2 for most of these functions.

Another extension from $\mathbb{R}$ to the intervals is the definition of comparison. Depending on the flavor, a comparison can have different meanings. Let us concentrate on the set-based flavor. There is no straightforward way to extend $\leq$ for instance: what is the result of $[1,4] \leq [2,3]$? It could be true as $1 \in [1,4]$ and $2 \in [2,3]$ and $1 \leq 2$, or false as $4 \in [1,4]$, $2 \in [2,3]$ and $4 \not\leq 2$. Based on Allen's work on temporal logic (Allen 1983), a list of predicates has been established. In our example, the predicate `containedIn`$([2,3],[1,4])$ holds. The other predicates are `bothEmpty`, `firstEmpty`, `secondEmpty`, `before`, `meets`, `overlaps`, `starts`, `finishes`, `equal`, `finishedBy`, `contains`, `startedBy`, `overlappedBy`, `metBy`, `after`. These predicates express all possible relative positions of the first interval with respect to the second. Using these predicates, the Boolean functions `equal`, `subset`, `less`, `precedes`, `interior`, `strictLess`, `strictPrecedes` and `disjoint` can be defined.

As intervals are sets, specific operations on sets are also required by the standard: the definition of `intersection` is straightforward, `convexHull` returns the tightest interval containing a union. Testing the inclusion is also possible. As intervals are specific sets, some more numeric non-arithmetic operations are required: `inf`, `sup` return the endpoints, `mid`, `wid`, `rad` return the midpoint, width and radius, `mag` and `mig` return the magnitude and mignitude.

An important class of applications of interval arithmetic consists in solving sets of constraints, that is, in finding values that satisfy simultaneously a set of equations or inequalities. In this respect, the so-called reverse operations are important operations; they are also known as relational operators. For instance, in $\mathbb{R}$, the set of solutions of $x^2 = 2$ is the set $\{-\sqrt{2}, \sqrt{2}\}$: it is not a real number but a set of numbers which are associated with the input value 2 through the square operation. This notion is extended to intervals. In the set-based flavor, the reverse of the following unary functions are required: ^{2}, $1/$, abs, pown, sin, cos, tan and cosh. For instance sinRev$([1/2,1],[0,\pi]$ is $[\pi/6, 5\pi/6]$: only solutions within the second argument are considered. For binary functions, two reverse functions are needed, one for each argument of the original binary function. The functions implicated are $+$, $-$, $*$, $/$, pow and atan, with a special definition for the reverse addition or subtraction.

Finally, inputs and outputs are defined at Levels 2 and 3. Inputs are done using <*type*> `−text2interval`, that converts a string into an interval using data of type *type* for its representation, where *type* can be for instance `binary64` to specify double precision floating-point numbers. Conversely, outputs are realized via a call to `interval2text` which returns a string.

The standard for interval arithmetic has to provide means to handle exceptions. To do so, a decoration is attached to each interval, describing a property of the operations and functions which produced this interval. Apart from the `com` decoration, where `com` stands for common and means that defined and continuous computations were performed on non-empty and bounded input intervals, the set of decorations and the rules to propagate decorations along with the computations are flavor-dependent. The set of decorations and the propagation rules for the set-based flavor are detailed in the draft text of the standard.

4 Main difficulties

The standardization of interval arithmetic is conducted mostly via e-mail. For some topics, rich and lively discussions took place before an agreement could be reached.

4.1 Issues at Level 1

At Level 1, one of the main difficulties was to choose the mathematical model. To start with, the notion of interval had to been agreed upon. The agreement on empty or not, bounded or unbounded intervals was easily reached. In the case an interval is unbounded, the question whether the infinity (either $-\infty$ or $+\infty$ or both) belongs to the interval has been answered negatively: otherwise $\{+\infty\}$ would be a valid interval. It was considered that an interval that contains solely an infinity is difficult to interpret and has to be avoided.

Flavors

A hot topic was the choice of the mathematical theory for interval arithmetic. No mathematical theory is universally superior to the other ones. Classical interval arithmetic (Moore 1966), where only non empty and bounded intervals are considered, was rapidly discarded because it lacks the power to express and handle useful intervals. For instance, Newton's algorithm can return an empty interval: this conveys the meaningful information that the problem has no solution. Set theory (Neumaier 1990, Jaulin et al. 2001, Moore, Kearfott and Cloud 2009, Tucker 2011) is the most famous theory and was favoured by some. However, proponents of Kaucher (Kaucher 1973, Kaucher 1980, Kulisch 2008) (or modal (Gardeñes et al. 1985)) arithmetic insisted on the importance of applications using this second theory. Finally, it was decided that the standard will accomodate several so-called *flavors*. Formally, a flavor is defined as an interpretation, or via an embedding, of intervals (common to all flavors) into the chosen mathematical theory.
This definition is elaborated so as to make it possible to accomodate a set-based flavor (which is entirely defined in the current draft of the standard), a Kaucher flavor (still pending) or a cset flavor (Pryce and Corliss 2006), that all return the same result when evaluating an expression which makes sense in each flavor.
With the set-based flavor, the embedding is straightforward, and the definition of an expression such as $\mathbf{y} = \varphi(\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_n})$ as the range of the point function φ on the sets $\mathbf{x_1}, \ldots \mathbf{x_n}$ is natural. In Kaucher flavor, entities are either bounded "direct" intervals, that is, intervals of the form $[\underline{a}, \bar{a}]$ with $\underline{a} \in \mathbb{R}$, $\bar{a} \in \mathbb{R}$ and $\underline{a} \leq \bar{a}$, or "reverse" intervals, that is, intervals of the form $[\bar{a}, \underline{a}]$ with $\underline{a} \in \mathbb{R}$, $\bar{a} \in \mathbb{R}$ and $\underline{a} \leq \bar{a}$. Unbounded intervals are not permitted. Operations and functions that are defined and continuous at each point of a direct interval $\mathbf{x} = (\mathbf{x_1}, \ldots \mathbf{x_n})$ behave as operations and functions of the set-based flavor. Furthermore, they are also defined for reverse intervals. Interpretations of direct and reverse intervals can be found in (Goldsztejn, Parts I and II, 2012). The cset flavor has been discussed several times but no motion concerning this flavor has been voted upon.

Other flavors can be proposed and considered for inclusion in the standard. The procedure to do so is detailed in the text of the standard, Annex B.

Decorations

Another hot topic was the handling of exceptions. Exceptions will occur and the mechanism to handle them had to be defined. We kept the principle, adopted in IEEE-754 standard for floating-point arithmetic, that a computation must go on in any case and must not halt abruptly. However, it was deemed necessary to keep track of exceptions, otherwise it would be possible to apply the theorem of Schauder incorrectly if "out-of-domain" was not recorded. The use of global flags, as in IEEE-754, has revealed weaknesses and was avoided. In particular, maintaining global flags seems to be inappropriate with multithreaded computing, as each thread would try to modify them and the semantics of the flags would be lost; in distributed computations (not even mentioning cloud computing), it would be extremely costly. In the standard for interval arithmetic, a flag, called *decoration* is attached to each interval. We are well aware that this solution also has drawbacks: it requires extra memory storage, ruins padding, may be a real pain if we want to reuse optimized floating-point routines, for instance if we want to implement linear algebra using floating-point BLAS.... A further difficulty was then to establish the list of possible flags and their propagation throughout the computation.

Finally, the following list (extracted verbatim from the draft standard version 8.1) of decorations for valid computations has been adopted, in descending order of strength:

Value	Short description	Definition
`com`	common	$\boldsymbol{x}$ is a bounded, nonempty subset of $\mathrm{dom}(f)$; f is continuous at each point of $\boldsymbol{x}$; and the computed interval $f(\boldsymbol{x})$ is bounded.
`dac`	defined & continuous	$\boldsymbol{x}$ is a nonempty subset of $\mathrm{dom}(f)$, and the restriction of f to $\boldsymbol{x}$ is continuous;
`def`	defined	$\boldsymbol{x}$ is a nonempty subset of $\mathrm{dom}(f)$;
`trv`	trivial	always true (so gives no information);

Propagation rules are rather natural. If we consider again an expression of the form $\mathbf{y} = \varphi(\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_n})$, the decoration of $\mathbf{y}$ depends whether φ is defined, continuous and bounded on $\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_n}$, but also on the decoration of each $\mathbf{x_i}$: the decoration of $\mathbf{y}$ is the minimum of these decorations, with respect to the order of strength. At Level 2, when implementing this sytem of decorations using a finite-precision type for the endpoints, one has to take care of overflows: the mathematical interval can be bounded but the represented one is unbounded because one of the endpoint could not be represented in the finite-precision type and an overflow occurred.

4.2 Issues at Level 2

Not an Interval

Exceptions are handled through decorations. The question whether "invalid" can appear is not entirely solved yet. As a corollary, at Level 2, the question whether NaI for "Not an Interval" should be introduced as a result has not been answered yet. For instance, computing $\sqrt{-2}$ is invalid and yields a NaN in floating-point arithmetic. But computing $\sqrt{[-2,-1]}$ is not invalid: it is simply $\sqrt{\emptyset}$, the square root of the interval $[-2,-1]$ intersected with the definition domain of square root, which is empty and perfectly valid. Invalid intervals can only appear during the construction of an interval, for instance if $[\texttt{bla}, 2]$ is given as an input for some function.

Representation of valid intervals

This is again a Level 2 issue. The most natural representation, found in every textbook about interval arithmetic (Moore 1966, Neumaier 1990, Moore, Baker and Cloud 2009…) is by endpoints. However, other representations such as mid-rad representations, namely by the midpoint and the radius, offer practical advantages, as exemplified by the product of interval matrices (Rump 1999). After close scrutiny it appears that only the representation by endpoints offers a unique representation of every interval: this representation is called *explicit*. The interval $[0, +\infty)$ has no mid-rad representation, the only way to enclose it using the mid-rad representation is to choose any nonnegative number as a midpoint and an infinite radius: it is represented *implicitely*. The standard requires at least a representation by endpoints.

Exact dot product

An exact dot product for vectors of floating-point numbers is a routine that acts as if it were computing the dot product exactly and rounding it before returning it. A recent hot topic has been whether the standard should require this routine. Tenants for the exact dot product insisted on the fact that interval computations deal with numerically guaranteed results. Exact dot product offers the guarantee that a dot product has the highest possible quality. At Level 3, tenants of the exact dot product also insisted on having it in hardware, for efficiency reasons, but eventually abandoned this idea, because it is too demanding for manufacturers, and it would jeopardize the wide adoption of this standard. Opponents to the exact dot product insisted on the fact that the standard for interval arithmetic should focus on interval computations and that the exact dot product is more related to floating-point issues. This last view has finally been adopted and the exact dot product is not required by the standard.

Accuracy and reproducibility

Accuracy and reproducibility have not been hot topics but are difficult to define properly in a standard. The first topic regards the numerical quality of the computed interval result, when interval arithmetic is implemented using floating-point arithmetic (Neumaier 2008). As the inclusion property must absolutely be preserved, the computed result must include the exact result: the coresponding accuracy mode is called *valid*. The ideal result would be the exact result, rounded outwards to preserve the inclusion property: the corresponding accuracy mode is called *tightest*. Tightest results may be reproducible. As tightest interval can be difficult to compute, an intermediate mode, called *accurate* has been defined. Loosely speaking, it asserts that the computed result is valid and is only slightly larger than the tightest accuracy mode applied to a slightly larger input interval.

5 What is still missing?

As of Version 8.1, 25 October 2013, remaining holes in the draft text of the standard concern flavors. Hooks have been provided in the text to accomodate several flavors. The set-based flavor is fully defined in Section 10, the corresponding section for the Kaucher flavor is still missing. It seems to me that the proponents for Kaucher arithmetic have abandoned the collective effort of the IEEE-P1788 working group to follow other tracks (N. Hayes September 2013).
The draft text needs wordsmithing and polishing: a few places have been identified where information must be more clearly distinguished from conformance requirements. They concern only minor details.
A still unresolved issue is the (present lack of a) reference implementation (Revol 2010). The group led by J. Wolff von Gudenberg (Nehmeier, Wolff von Gudenberg and Pryce 2013) is developing such a reference fully conforming implementation. Another effort is conducted by D. Nadezhin and S. Zhilin (Nadezhin, and Zhilin 2012). Experimenting using this reference implementation will help to check the validity of the choices made for the elaboration of this standard. It will also help to identify places where clarification is needed.
The last, still missing, step is the development of a test suite: what should a test suite contain in order to test every cornercase, every possibility? to test tightness?

6 Conclusion

Eight months remain before this talk is given at the ICVRAM 2014 conference. I hope to be able to present a more complete panorama of the future IEEE 1788 standard for interval arithmetic, and to exhibit a quasi-final version of the corresponding text.

References

[1] J. F. Allen (1983). *Maintaining knowledge about temporal intervals*, Communications of the ACM, vol. 26, pp. 832–843.

[2] W. Edmonson, and G. Melquiond (2009). *IEEE Interval Standard Working Group-P1788: Current Status*, ARITH 2009: 19th IEEE Symposium in Computer Arithmetic, Portland, USA, pp. 231–234.

[3] E. Gardeñes, H. Mielgo and A. Trepat (1985). *Modal Intervals: Reason and Ground Semantics*, Interval Mathematics, Lecture Notes in Computer Science, vol. 212, pp. 2735.

[4] A. Goldsztejn (2012). *Modal Intervals Revisited, Part 1: A Generalized Interval Natural Extension*, Reliable Computing, vol. 16, pp. 130-183.

[5] A. Goldsztejn (2012). *Modal Intervals Revisited, Part 2: A Generalized Interval Mean Value Extension*, Reliable Computing, vol. 16, pp. 184-209.

[6] N. Hayes, *That other flavour...*, message to the IEEE P1788 mailing list, 21 September 2013, `http://grouper.ieee.org/groups/1788/email/msg07157.html`

[7] Institute of Electrical and Electronic Engineers (1985). and American National Standards Institute. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard, Std 754-1985,* New York.

[8] Institute of Electrical and Electronic Engineers (2008). IEEE standard for floating-point arithmetic. *IEEE Standard 754-2008,* New York.

[9] IEEE Interval Standard Working Group - P1788: online resources `http://grouper.ieee.org/groups/1788/`

[10] L. Jaulin, O. Didrit, M. Kieffer, and E. Walter (2001). *Applied Interval Analysis*, Springer.

[11] E. Kaucher (1973). *Über metrische und algebraische Eigenschaften einiger beim numerischen Rechnen auftretender Raume*, PhD thesis, Karlsruhe, Germany.

[12] E. Kaucher (1980). *Interval Analysis in the Extended Interval Space* $I\mathbb{R}$, Computing, Suppl., vol. 2, pp. 3349.

[13] R. B. Kearfott, J. Pryce and N. Revol (2009). *Discussions on an Interval Arithmetic Standard at Dagstuhl Seminar 08021*, follow-up of the seminary on *Numerical Validation in Current Hardware Architectures*, at Dagstuhl, Germany, January, 2008. Lecture Notes in Computer Science no 5492, pp 1–6.

[14] U. Kulisch (2008). *Computer Arithmetic and Validity*, de Gruyter.

[15] R. E. Moore (1966). *Interval analysis*, Vol. 2, Englewood Cliffs: Prentice-Hall.

[16] R. E. Moore, R. B. Kearfott, and M. J. Cloud (2009). *Introduction to Interval Analysis*, SIAM.

[17] D. Nadezhin, and S. Zhilin (2012). *JInterval library: principles, development, and perspectives*, SCAN 2010: 15th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Verified Numerical Computations, Novosibirsk, Russia.

[18] M. Nehmeier, J. Wolff von Gudenberg, and J. Pryce (2013). *On Implementing the IEEE Interval Standard P1788*, FEMTEC 2013, Las Vegas, USA, p. 103.

[19] A. Neumaier (1990). *Interval Methods for Systems of Equations*, Cambridge University Press.

[20] A. Neumaier (2008). *Vienna proposal for interval standardization*, `http://www.mat.univie.ac.at/~neum/ms/1788.pdf`

[21] J. D. Pryce, and G. F. Corliss (2006). *Interval arithmetic with containment sets*, Computing, vol. 78, pp. 251–276.

[22] N. Revol, (2010). *Standardized interval arithmetic and interval arithmetic used in libraries*, ICMS 2010: International Conference on Mathematical Software, Kobe, Japan, pp. 337–341.

[23] S. M. Rump (1999). INTLAB - INTerval LABoratory, In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77-104. Kluwer Academic Publishers, Dordrecht.

[24] W. Tucker (2011). *Validated Numerics*, Princeton University Press.