The MPFI library revisited

Nathalie Revol*[1]

[1]University of Lyon - Inria, France

Keywords: MPFI Library; Arbitrary Precision Interval Arithmetic; IEEE 1788-2015 Compliance

The IEEE 1788-2015 standard

Interval arithmetic has been defined and used since the 50s and 60s. However, no common definition existed for years and it made difficult to compare different works. In 2008, a group of interval experts, gathered at a seminar in Dagstuhl, felt that interval arithmetic was mature enough to undergo a standardization effort. This effort led to the IEEE 1788-2015 standard [2].

It was impossible to define a theory that encompasses the co-existing theories in use, such as set theory, Kaucher arithmetic, modal arithmetic, cset arithmetic. The adopted solution was to provide "hooks" to accomodate different theories within the standard: each provided theory is called a *flavor*. The only flavor defined in the 2015 version of the standard is the set-based flavor, from set theory.

Another peculiarity of the IEEE 1788-2015 standard is the handling of exceptions, called decorations. A *decoration* is attached to each interval and gives a summary of what happened during the computations that resulted in this interval: was every operation defined and continuous over its arguments, or simply defined, or even less, such as in $\sqrt{[-2,1]}$ where the square root is not defined everywhere over its argument $[-2,1]$? Incidentally, in the set-based flavor, $\sqrt{[-2,1]}$ is computed as $\sqrt{[-2,1] \cap \mathrm{Dom}_{\sqrt{}}} = \sqrt{[0,1]} = [0,1]$.

*Corresponding author.

Libraries compliant with the IEEE 1788-2015 standard

The development of the standard has been accompanied by the development of the C++ `libieee1788` library by M. Nehmeier, that served as a proof-of-concept. Unfortunately, M. Nehmeier left academia and this library is no more maintained. Two other libraries have been developed since then and are compliant with the standard: `JInterval` by D. Nadezhin and S. Zhilin, and the Octave `interval` package by O. Heimlich. The `JInterval` recently and untimely lost its main developer, D. Nadezhin. O. Heimlich also left academia but he still develops and maintains the Octave `interval` package.

No other library of interval arithmetic has been developed in compliance with the IEEE 1788-2015 standard, because it is difficult. A first difficulty is the implementation of the long list of functions and conversions mandated by the standard, with the prescribed accuracy. Another difficulty is the implementation of the decoration mechanism. On the one hand, it requires that an extra piece of information is attached to each interval, and this can destroy memory optimizations (padding etc.). On the other hand, decorations must be propagated and this implies some more extra code. These difficulties are less salient for the MPFI library, introduced below.

The MPFI library

MPFI [3] is a C library for arbitrary precision interval arithmetic. An interval is represented by its endpoints, which are arbitrary precision floating-point numbers provided by the MPFR library [1]. Every single operation is

as accurate as possible, thanks to the MPFI library that provides correctly rounded operations, with directed roundings as needed, for each endpoint.

MPFR already offers a long list of functions and conversions between different types and MPFR floating-point numbers: incorporating them in MPFI is usually relatively easy, for most of them, as the bulk of the work has already been done by MPFR developers. However, some functions mandated by the standard, and in particular most of the *reverse* functions, useful for constraints solving, are still missing in MPFI.

As an interval is represented by two arbitrary, and thus variable, precision endpoints, adding a decoration to each interval is not an issue: padding or cache optimization are not at stake anyway, as the employed memory is already (usually) variable and large.

Finally, the mechanism for handling exceptions in MPFI is very different from the one adopted in the IEEE 1788-2015 standard: for instance, for $\sqrt{[-2,1]}$, MPFI returns `NaI`, which stands for `Not an Interval`. The code of each MPFI operation must be reworked to handle and propagate decorations.

To sum up, there is some work to be done to make MPFI compliant with the IEEE 1788-2015 standard, but this work seems less demanding than for libraries based on fixed-precision floating-point numbers such as IEEE 754-2008 `binary32` or `binary64`. The relative overhead, both in terms of memory and of computation time, due to the incorporation of flavors and decorations, is also less important and probably negligible.

Work to be done

The main modifications will take place at two levels. The first one concerns the data structure of a MPFI interval.

a) An extra field will be added to indicate the flavor in use. This is a bit different (but not incompatible) from the intended use of a flavor, which is supposed to be set for a whole block of code rather than for an individual interval. However, MPFI will check that the flavors of the operands and of the result match before performing the required operation.

b) An extra field, parameterized by the flavor, will be added to store the decoration attached to the interval.

The second kind of modifications concerns the code for each operation.

a) A preprocessing will be added to check the compatibility of the flavors and to branch to the code corresponding to the flavor in use.

b) For each branch, a postprocessing will propagate the decoration.

Lastly, for backward compatibility, a `MPFIoriginal` flavor will be added, that will branch to the original version of MPFI. It will be the default flavor, so that users can run their existing codes without any modification of their behaviour.

References

[1] F. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding. *ACM Transactions on Mathematical Software*, 33(2):no 13, 2007.

[2] IEEE: Institute of Electrical and Electronic Engineers. IEEE 1788-2015 Standard for Interval Arithmetic, 2015.

[3] N. Revol and F. Rouillier. Motivations for an Arbitrary Precision Interval Arithmetic and the MPFI Library. *Reliable Computing*, 11(4):275–290, 2005.