Computations on symbolic floating point numbers

C.-P. Jeannerod, N. Louvet, J.-M. Muller, **A. Plet**
Département, Laboratoire de l'Informatique du Parallélisme
46 Allée d'Italie, Lyon 69364
antoine.plet@ens-lyon.fr

November 3-7, 2014

In a computer, real numbers are often approximated by a finite discrete set called floating point numbers. Therefore, any computation leads to an error we have to care about. When it comes to compute successive basic operations, a naive algorithm can lead to a huge relative error on the result. For example, for an expression as simple as a 2x2 determinant, you can get a relative error bigger than 1 which is not acceptable. Any algorithm computing over floating point numbers should be given with a bound on the possible output error. Such a proof of accuracy can sometimes be independent from the precision or from the rounding scheme, then covering various standard formats defined in IEEE754-2008 [1].

The next step to achieve a complete analysis of an algorithm is to provide an optimal error bound. Testing the optimality for all the possible inputs does not scale with the increasing precision we are able to compute and the optimality of the error bound in [3] comes from a generic example, parametrized with the precision. Let's call it a symbolic floating point number. However, the computations on such generic examples are done by hand which is time demending and error prone. We propose two methods to compute additions and multiplications followed by a rounding operation on symbolic floating point numbers. Those methods are valid for a subset of floating point numbers which could be called sparse symbolic floating point numbers and for the default rounding scheme "round to nearest ties to even".

The first method relies on the evaluation-interpolation scheme available for polynomial computations. Sparse symbolic floating point numbers can be

represented as a polynomial. The dependency on the precision goes all to the evaluation point and none to the polynomial. That is, for every possible precision, the floating point number can be seen as the evaluation of the same polynomial at a different point. Of course, when we add or multiply such numbers with one another, without any rounding operation, we keep the same property. The interesting and non trivial fact is that for the default rounding scheme, the polynomial point of view is still valid after the rounding operation : the result is a new polynomial, evaluated in the same symbolic value. Thus, one can compute the result of a symbolic floating point operation from numerical evaluations of the operation for various precisions and then get the results back to the symbolic word.

The second method is inspired from the natural way to compute on floating point numbers. We first locate the most significant digit and deduce the position of the least significant digit. Then we can decide how to truncate and compensate the initial number to get the right result according to the rounding scheme.

We implemented those two methods which allowed us to instantly check the available examples in [2], [3], [4].

References

[1] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*, Available at http ://ieeexplore.ieee.org/servlet/opac ?punumber=4610933

[2] R. Brent, C. Percival, P. Zimmermann, *Error Bounds on Complex Floating-Point Multiplication*, Mathematics of Computation 76 (2007), pp. 1469–1481

[3] C.-P. Jeannerod, N. Louvet, J.-M. Muller, *On the componentwise accuracy of complex floating-point division with an FMA*, Proceedings of the 21st IEEE Symposium on Computer Arithmetic (2013), pp. 83–93

[4] C.-P. Jeannerod, N. Louvet, J.-M. Muller, *Further analysis of Kahan's algorithm for the accurate computation of* 2×2 *determinants*, Mathematics of Computation 82 (2013), pp. 2245–2264