Postdoctoral position: ProvenMPFR

Contact: Paul Zimmermann, `Paul.Zimmermann@inria.fr`

Location: project-team Caramba, Inria Nancy - Grand Est, Nancy, France

Dates: 18 months, starting up from November 1st, 2019.

Salary: 2653 euros gross/month (before taxes).

The aim of this project is to formally prove the low-level routines of the GNU MPFR library, which is used — among other software tools — by the compiler suite GCC, the CGAL computational geometry library, the Gnome calculator, the Julia language, the computer algebra systems Maple and SageMath, the Matlab multiple-precision toolbox, the GNU Octave software tool.

When the Pentium FDIV bug was discovered in 1994, Intel hired John Harrison, a specialist of formal proof, to prove the conformance of its next-generation processors. Instead of waiting for a critical bug to be found in GNU MPFR, the long-term objective is to formally prove the correctness of the MPFR library. We want not only to formally prove the algorithms used, but also their implementation in the C language. The goal of the ProvenMPFR postdoctoral project is to investigate on the low-level MPFR routines, and determine the best tools to realize that project.

Several authors have worked in the last years on the formal proof of floating-point algorithms and the IEEE-754 standard, in particular [5, 4]. However, only a few authors have considered *arbitrary precision* floating-point algorithms (in the case of integers we can mention [1], [2] for decimals of π, and the on-going work of Raphaël Rieu-Helft in his PhD thesis[1]).

One of the difficulties of arbitrary precision is that numbers are stored on several machine words. For example, a 1000-bit floating-point number can be represented on 16 words of 64 bits each, the last word containing only 40 significant bits. We thus have essentially two layers: a logical layer of n-bit numbers, and a physical layer where those numbers are split into machine words. For efficiency reasons, arbitrary-precision floating-point implementations use both operations on the logical layer (for example to multiply two arbitrary-precision numbers) and operations on the physical layer (for example to correctly round a floating-point number).

In parallel, some tools do appear for the formal verification of low-level programs [3]. Norbert Müller (Trier) has started working towards the formal proof of its RealRam implementation [7] (which is also based on MPFR).

The MPFR library is a library written in the C language, which performs computations on arbitrary-precision floating-point numbers, and guarantees *correct rounding*. Defined by the IEEE-754 standard, correct rounding allows to specify in a unique way the result of a floating-point

[1] `https://www.lri.fr/~rieu`

operation (once a rounding mode is given). This leads to deterministic — and thus reproducible — computations, whatever the processor used, the compiler, and the operating system. Published open-source since 2000, MPFR implements an extension of the IEEE-754 standard in arbitrary precision, extension which has been itself standardized by the 2008 revision of the standard. The algorithms used by MPFR and the corresponding error bounds are published in the `algorithms.pdf` document available on the MPFR web page[2].

The ProvenMPFR project will explore the formal proof of algorithms computing correctly-rounded results for arbitrary-precision floating-point numbers. Instead of proving algorithms with "pen and pencil", the goal is to prove the concrete implementation of those algorithms in MPFR. We will mostly aim at the basic arithmetic routines (addition, subtraction, multiplication, division, square root) for precisions up to 128 bits[3]. The article [6] demonstrates how we can split those operations in two steps: first the computation of an approximate value, then the determination of the correct rounding. The goal of the ProvenMPFR project is:

1. to explore the different tools currently available, that would allow to formally prove the algorithms used by MPFR, *and* their implementation in the C language;

2. instrument at least two of these tools to prove the correctness of the basic MPFR routines in small precision, as described in [6]. In particular, we should consider at least a tool doing code extraction, and a tool working with code annotation. In the case of code extraction, the extracted code should be as efficient as that currently used by the MPFR library, so that we can replace it by the extracted code. In the case of code annotation, modifying the MPFR code is allowed, as long as its efficiency remains the same.

Some preliminary work was done in 2018 by Jianyang Pan during his Master intership, where he was able to prove the correctness of the 1-word addition in GNU MPFR (and partially for the 1-word multiplication); the corresponding extracted code is as efficient as the original code, and is already available in MPFR, when configured with `-enable-formally-proven-code`. The MPFR library being evolving continuously, that comparison should also take into account how easy it will be for the MPFR developers — who are not fluent in formal proof — to learn and use those tools.

Funding. The funding of this postdoctoral position is secured by Inria's PRE ("Projet de Recherche Exploratoire") ProvenMPFR.

Context and Followup Work. The hired researcher will work in the Caramba team in Nancy, and will mainly interact with the two main MPFR developers, namely Paul Zimmermann in the Caramba team, and Vincent Lefèvre in the AriC team (Lyon, France). It is expected that this postdoctoral position will yield scientific publications in the main conferences/journals of formal proof and/or computer arithmetic. Based on those scientific results, an ERC (European Research Council) Advanced Grant project will be submitted. If successful, the researcher hired on this postdoctoral position will be of course a natural candidate for the ERC funding.

[2] `http://www.mpfr.org/algorithms.pdf`

[3] The IEEE-754 standard allows of course larger precisions, but the MPFR library uses some optimized code for small precisions. This code should be easier to prove than the generic code used for larger precisions.

Prerequisites. A PhD in computer science is required. This postdoctoral position requires some experience in the concrete application of one or more formal proof tools, with published articles in that domain. A good knowledge of the C language, beyond small programs, is required, since it will be necessary to read and understand the C code used by MPFR, and if needed modify it to make the proof easier, but without any loss of efficiency. Some knowledge of computer arithmetic (IEEE 754 standard, arbitrary precision) is welcome but not required.

References

[1] Bertot, Y., Magaud, N., and Zimmermann, P. A proof of GMP square root. *Journal of Automated Reasoning 29* (2002), 225–252. Special Issue on Automating and Mechanising Mathematics: In honour of N.G. de Bruijn.

[2] Bertot, Y., Rideau, L., and Théry, L. Distant decimals of π. *Journal of Automated Reasoning* (2017), 1–45.

[3] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Hritcu, C., Protzenko, J., Ramananandro, T., Rastogi, A., Swamy, N., Wang, P., Béguelin, S. Z., and Zinzindohoué, J. K. Verified low-level programming embedded in F^*. *CoRR abs/1703.00053* (2017).

[4] Boldo, S., and Melquiond, G. *Computer Arithmetic and Formal Proofs.* ISTE Press - Elsevier, Dec. 2017.

[5] Daumas, M., Rideau, L., and Thery, L. A Generic Library for Floating-Point Numbers and Its Application to Exact Computing. In *Theorem Proving in Higher Order Logics* (Edinburgh, United Kingdom, 2001), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 169–184.

[6] Lefèvre, V., and Zimmermann, P. Optimized Binary64 and Binary128 Arithmetic with GNU MPFR. In *24th IEEE Symposium on Computer Arithmetic (ARITH 24)* (London, United Kingdom, July 2017).

[7] Müller, N. T., and Uhrhan, C. Some steps into verification of exact real arithmetic. In *Proceedings of the 4th NASA Formal Methods Symposium (NFM 2012)* (Berlin, Heidelberg, April 2012), A. E. Goodloe and S. Person, Eds., vol. 7226, Springer-Verlag, pp. 168–173.