Sollya: an environment for the development of numerical codes

Sylvain Chevillard, Mioara Maria Joldes, Christoph Lauter

Sollya: an environment for the development of numerical codes

S. Chevillard[1], M. Joldeş[2], and Ch. Lauter[2]

[1] INRIA, LORIA, Caramel Project-Team,
BP 239, 54506 Vandœuvre-lès-Nancy Cedex, FRANCE
[2] LIP (CNRS/ÉNS de Lyon/INRIA/Université de Lyon), Arénaire Project-Team,
46, allée d'Italie, 69364 Lyon Cedex 07, FRANCE**

Abstract. Sollya has become a mature tool for the development of numerical software. With about 175 built-in algorithms and a broad extensibility, it offers a complete tool-chain for fixed- and floating-point software and hardware design. Its features include on-the-fly faithful rounding, specialized approximation algorithms and extensive support for floating-point code generation.

Keywords: Numerical software, faithful rounding, computer algebra, development tool, function approximation

1 Introduction

The software tool Sollya is an interactive environment assisting developers of numerical codes in the design of mathematical routines. It provides a safe and fast experiment bench as well as tools to generate—at least partially—such codes. The users can either use it through an interactive shell (where they enter commands and get results from the tool) or as a scripting language. The language is intended to make the manipulation of numerical expressions particularly easy.

Initially, Sollya was intended more specifically for people implementing numerical functions in mathematical libraries (these functions are, e.g., exp, arccos, tanh, etc.). Since then, the tool has evolved and has now become interesting not only to developers of mathematical libraries, but also to everyone who needs to perform numerical experiments in an environment that is safe with respect to round-off errors. Recently, it has even been used for more ambitious projects, such as MetaLibm[3].

Sollya is a free software distributed under the terms of the CeCILL-C license and available at `http://sollya.gforge.inria.fr/`. The current version is Sollya 2.0, released in April 2010. A complete documentation with tutorials is available [3], and Sollya integrates an interactive help system.

** Ch. Lauter is now at Intel Corporation, 2111 NE 25th Avenue, M/S JF1-13, Hillsboro, OR, 97124, USA, but he was in the Arénaire Project-Team when he developed Sollya.

[3] See `http://lipforge.ens-lyon.fr/www/metalibm/`.

2 Context and competing tools

The development of an arithmetical operator, such as a mathematical function f (e.g., $f = \log$) or a conversion operator, usually steps through four phases. In each of these phases, specific questions are addressed. Examples are given below. They can be answered using Sollya.

- Analysis phase: How to classify the numerical behavior of function f? Does it allow for any kind of range reduction (see, e.g., [10, Chapter 11])?
- Approximation phase: How to compute a polynomial p with minimal degree approximating f such that the approximation error stays below some bound?
- Code generation phase: How can an approximation polynomial p be implemented with bounded round-off error?
- Validation phase: What is a safe yet tight bound for the approximation error of p with respect to f? What are appropriate test vectors that exercise the IEEE 754 properties of the operator, such as flag settings [1]?

Most algorithms offered by Sollya are also supported by other tools such as Maple, Mathematica or Matlab. However, when used for the development of numerical codes, these tools show several deficiencies [6]. As their focus is more on general computer algebra, some of their numerical algorithms are not optimized for performance and support for floating- or fixed-point arithmetic is limited. Most importantly, while they generally offer support for arbitrary precision, the actual *accuracy* of computations in these tools is often unknown.

3 Key features offered by Sollya

Sollya focuses on providing arbitrary accuracy for function evaluation through on-the-fly faithful rounding. More precisely, the user may define univariate functions as expressions made up of basic functions, such as exp, sin, etc. Sollya can evaluate such function expressions at points providing results with the accuracy for which the user asked. If the working precision needs to be adapted to achieve that accuracy, Sollya takes the burden of doing so off the user.

In this process, Sollya makes sure it never lies: if ever the tool is not able to exhibit a faithful rounding or if rounding might flip the result of a comparison, it will warn the user of that problem with the result. This rigor is achieved through an extensive use of Interval Arithmetic [9], extended to cover functions with false singularities [4]. Interval Arithmetic is of course available at the Sollya interface, too.

Sollya currently supports about 175 commands in a scripting language that enables structured programming. The tool can be extended through dynamically loaded plug-ins. It is out of the scope of this paper to cover this wide range of functionalities. We shall hence concentrate on four outstanding Sollya features.

- Polynomial approximation in sparse monomial bases: Given a function f and a bounded domain I, Sollya can compute an approximation polynomial of

least error in some monomial basis which, in contrast to other tools, may be sparse. So the polynomial may be, e.g., of the form $p(x) = a_2\,x^2 + a_3\,x^3 + a_7\,x^7$. This helps with reducing the number of operations needed to evaluate such approximation polynomials [7].
- Taylor Models of univariate functions: Given a function and a bounded domain I, Sollya can compute a Taylor expansion p of f together with a rigorous interval bound Δ enclosing the error $p(x) - f(x)$ for all $x \in I$. Such Taylor Models allow the behavior of f to be rigorously analyzed. They also help with problems like validated integration [8].
- Supremum norms of approximation error functions: Sollya can compute safe bounds on the supremum norm of the error $\varepsilon = p/f - 1$ made when replacing a function f by a particular polynomial p in a bounded domain I. Such computations are a requirement for code certification for mathematical functions [4].
- Support for code generation in IEEE 754 arithmetic: Sollya offers extensive support for simulating IEEE 754 arithmetic. Additional commands enable the generation of IEEE 754-based C code with bounded round-off error [6].

4 Conclusion and project future

Sollya aims at providing a safe environment for numerical computations. Its main feature in comparison to competing tools such as Maple is the fact that the tool always tries to guarantee the numerical quality of the results by giving explicit error bounds.

Sollya is still under development. However it has already been used for several software projects involved with floating-point arithmetic: it has been used for the development of large parts of the CRLibm library[4] [6] and, more recently, for the FLIP library[5] [5]. Several research teams are using the tool for ongoing research (e.g., [2]). Finally, Sollya has been reported (in a private communication) to be used for research and development in industry, where it covers both the hardware as the software side of development in computer arithmetic.

In the future, the following features could be added. First, commands like numerical integration or polynomial interpolation are currently missing. Second, due to its history, Sollya only handles univariate real functions. Originally this choice was reasonable as the focus was on the development of mathematical libraries. But now that a larger community is targeted, support for multivariate functions would be interesting. Complex arithmetic could also be added, with, in particular, commands for computing rational and polynomial best approximations in the complex plane. Finally, support for linear algebra could be added in the long term: algorithms for product and inverse of matrices, linear system solvers, eigensolvers, etc. To follow the Sollya philosophy, these implementations would also adapt their working precision automatically to guarantee the correctness and accuracy of the results.

[4] See `http://lipforge.ens-lyon.fr/www/crlibm/`.
[5] See `http://flip.gforge.inria.fr/`.

References

1. American National Standards Institute (ANSI) and Institute of Electrical and Electronic Engineers (IEEE). IEEE Standard for Binary Floating-Point Arithmetic (IEEE Std754-2008), 2008. Revised version of the IEEE Std754-1985 Standard.
2. M. G. Arnold, S. Collange, and D. Defour. Implementing LNS using filtering units of GPUs. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2010*. `http://hal.archives-ouvertes.fr/hal-00423434`.
3. S. Chevillard, M. Joldeş, and Ch. Lauter. User's Manual of the Sollya Tool, Release 2.0. Available at `http://sollya.gforge.inria.fr/`.
4. S. Chevillard and Ch. Lauter. A certified infinite norm for the implementation of elementary functions. In *Proceedings of the Seventh International Conference on Quality Software*, pages 153–160, 2007.
5. C.-P. Jeannerod, H. Knochel, C. Monat, G. Revy, and G. Villard. A new binary floating-point division algorithm and its software implementation on the ST231 processor. In *Proceedings of the 19th IEEE Symposium on Computer Arithmetic*, pages 95–103, Portland, OR, USA, June 2009.
6. Ch. Lauter. *Arrondi correct de fonctions mathématiques. Fonctions univariées et bivariées, certification et automatisation.* PhD thesis, École Normale Superieure de Lyon, Université de Lyon, 2008.
7. Ch. Lauter and F. de Dinechin. Optimizing polynomials for floating-point implementation. In *Proceedings of the 8th Conference on Real Numbers and Computers*, pages 7–16, Santiago de Compostela, Spain, July 2008.
8. K. Makino and M. Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 4(4):379–456, 2003. `http://bt.pa.msu.edu/pub/papers/TMIJPAM03/TMIJPAM03.pdf`.
9. Ramon E. Moore. *Methods and Applications of Interval Analysis.* Society for Industrial Mathematics, 1979.
10. J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic.* Birkhäuser Boston, 2009. `http://www.springer.com/birkhauser/mathematics/book/978-0-8176-4704-9`.