Sardana: an Automatic Tool for Numerical Accuracy Optimization

Ioualalen Arnault[1,2,3], Martel Matthieu[1,2,3]

[1]Univ. Perpignan Via Domitia, Digits, Architectures et Logiciels Informatiques, F-66860, Perpignan, France
[2]Univ. Montpellier II, Laboratoire d'Informatique Robotique et de Microélectronique de Montpellier, UMR 5506, F-34095, Montpellier, France
[3]CNRS, Laboratoire d'Informatique Robotique et de Microélectronique de Montpellier, UMR 5506, F-34095, Montpellier, France
arnault.ioualalen@univ-perp.fr, matthieu.martel@univ-perp.fr

Keywords: numerical accuracy, abstract interpretation, code synthesis.

On computers real numbers are approximated by floating-point numbers defined by the IEEE754 formats [1]. For most computations these formats are precise enough even though the induced approximation errors inherently. However in some cases the accuracy of the calculation is critical and the user needs to certify that his program will always yield an accurate enough output for every valid input. As it is impossible to check the validity of a calculation for every inputs, static analyzers such as Fluctuat [4] or Astrée [2] rely on an interval or relational representation of the inputs, combined to abstract interpretation.
Sardana is a tool designed to automatically rewrite numerical computations performed in floating-point arithmetics in order to optimize their accuracy. Sardana works directly on the source code of a LUSTRE program such as the ones used in real avionic software and produces a new source code as well as an absolute bound of error which is less than the original one. To achieve this Sardana uses: (*i*) Interval analysis, to handle large sets of inputs and not only isolated traces, (*ii*) A novel intermediate representation of program called APEG [5] which allows us to manipulate many transformed versions of the initial program in a compact way, (*iii*) A local search heuristic [5] to synthesize from an APEG a new version of the program, (*iv*) And abstract interpretation [3] to enforce the validity of our analysis of the accuracy.

The first challenge is how to transform a program into a more accurate one. As there is an exponential number of ways to write an arithmetic expression (e.g. a simple sum of n terms), we cannot exhaustively generate all possible transformations. This problem is closely related to the *phase ordering problem* of compilers. We use abstract interpretation to narrow down this search space

while allowing to represent in an abstract way as many transformed versions as possible of the initial program. Our structure of Abstract Program Expression Graph (APEG) is built from the syntactic tree of the source code, and is a compact and efficient way to handle multiple versions of a program without duplication and exponential growth of the structure. As there are many transformations which involve only the same part of the program, APEGs merge them locally into one equivalence class without duplicating the rest of the structure. Also, we introduce the concept of abstraction boxes into APEGs, which are defined by an operator and a set of sub-expressions. Each abstraction box allows to represent the exponential number of expressions that can be synthesized with the given operator over the set of sub-expressions of the box.

Next, Sardana has to extract from an APEG a program which has a better numerical accuracy. We use a limited depth search strategy with memoization. Intuitively we select the best way to evaluate an expression by considering only the best way to evaluate its sub-expressions. To accurately calculate both rounding errors and floating-point values, Sardana uses the GMP and MPFR libraries. Sardana is also able to manipulate any floating-point IEE754 format and fixed-point arithmetic.
Several experimental results have been obtained on various benchmarks and real-case applications, such as: summation (results are 50% close to the real values), polynomial functions like Taylor expansions (20% to 30% more accurate), and real avionic codes (10% more accurate half the time). Finally, Sardana provides a graphical interface allowing the user to specify the analyzer parameters easily and analyze the results in a user friendly way.

References:

[1] ANSI/IEEE, *IEEE Standard for Binary Floating-point Arithmetic*, Std 754-2008 edition, 2008.

[2] J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival, Static Analysis and Verification of Aerospace Software by Abstract Interpretation, AIAA Infotech@Aerospace, 2010.

[3] P. Cousot and R. Cousot, Abstract Interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixed points, *Principles of Programming Languages*, pp. 238–252, 1977.

[4] D. Delmas, E. Goubault, S. Putot, J. Souyris , K. Tekkal , and F. Vedrine, Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software, *Formal Methods for Industrial Critical Systems (FMICS)*, 2009.

[5] A. Ioualalen and M. Martel, A New Abstract Domain for the Representation of Mathematically Equivalent Expressions, *Static Analysis Symposium (SAS)*, 2012.