Automatic Analysis of Imprecision Errors in Software

E. Goubault and S. Putot

CEA/LIST, MeASI, 91191 Gif-sur-Yvette, FRANCE

Machines only compute with finite-precision arithmetic, using for instance IEEE 754 floating-point number, an approximation of the mathematical real number field. Also, data that programs process are not always known with precision. There are hopefully a number of means to deal with this: a careful mathematical design of the algorithmics involved, the use of alternative arithmetics, the study of the effect on the algorithm of this imprecision, and of uncertainties of its inputs and parameters. For the last point, different methods have been designed, among which interval and stochastic methods. Our approach is slightly different [4]. We start with a source code written in C, assertions describing the range of inputs, their imprecision errors, and possibly the range of the gradient of their values, and we automatically compute, i.e. by a tool, an over-approximation of the range of all variables known at the end of the code, and of their imprecision errors. The latter have two sources: the propagation of the initial imprecision errors, and the errors due to the IEEE 754 semantics of floating-point computations. This is based on abstract interpretation [3], which ensures the correctness of the approach. We only describe in this abstract some of the basic ideas of the underlying mathematical model computing these over-approximations, the full model will be developped in the final version of the paper.

Our mathematical model describing the propagation of errors for sets of executions of a code is based on ideas from affine arithmetic [2]. Affine arithmetic is an improvement of interval arithmetic allowing to take into account linear correlations. Typically, if x takes its values in [0,1], $x - x$ computed in interval arithmetic is [-1,1]: the information that the two x in this expression have the same value in [0,1] is lost. In affine arithmetic, a noise symbol ε_i, lying in [-1,1] is introduced to represent the fact that x is not known exactly. The sharing of noise symbols between variables expresses correlation. We write here $\hat{x} = 0.5 + 0.5\varepsilon_i$. Then $\hat{x} - \hat{x}$ is found to be zero.

However, affine arithmetic is only modeling real numbers, not floating-point numbers. In the model we propose, we use it both to over-approximate the result of the computation in real numbers, and the error between the real result and the floating-point result.

Moreover, we want to indicate, at the end of a computation, the source of errors in the program: for that, in the decomposition of errors using affine arithmetic, the terms have a meaning (which will be detailed in the following) and are not only used for expressing correlations.

Let x be the variable resulting from a computation, f^x an over-approximation of the set of floating-point values resulting from this computation, r^x an over-approximation of the set of real values resulting from the same computation in real numbers, and e^x an over-approximation of the error committed between the floating-point and the real computations. Moreover, we decompose the error onto a first-order error term, in which we want to detail contributions from all operations, and a higher-order error term, which is most of the time negligeable, but which we still want to bound, although without detailing all sources of errors. We write

$$f^x = r^x + e_1^x + e_{ho}^x$$

At each operation corresponds a label. Some of these operations introduce uncertainties on the value (for example an input given in an interval, or a non linear operation), we note them $i \in I$: a noise symbol $\varepsilon_i \in [-1, 1]$ is created for each of these labels, and the sets of real values taken by x is expressed as a sum

$$r^x = \alpha_0^x + \bigoplus_{i \in I} \alpha_i^x \, \varepsilon_i,$$

where the α_i^x coefficients express the propagation on x of the initial uncertainty introduced at point i.

Some operations (most of them) introduce new rounding errors, we note them $l \in L$. Of course, I and L are in general not disjoint. The rounding error committed at some operations can be computed very accurately, this is the case when the floating-point result of this operation is known exactly (i.e. not in an interval), these points are noted by $l \in L_1$, and no noise symbol needs to be created. In other cases, the new rounding error is bounded by a centered interval, which can easily be rewritten as a new noise symbol η_l multiplied by a real coefficient, these points are denoted $l \in L_2$. We then write the first order error

$$e_1^x = \bigoplus_{l \in L_1} t_l^x + \bigoplus_{l \in L_2} {t'}_l^x \, \eta_l + \bigoplus_{i \in I} {t''}_i^x \, \varepsilon_i + \beta_0^x + \bigoplus_{p \in P} \beta_p^x \, \vartheta_p$$

In this expression, the terms t_l^x and ${t'}_l^x \, \eta_l$ have already been described, they represent the first-order error associated to operation l. The other terms are useful for modelling the propagation of first-order error terms after a multiplication, when an error has been multiplied by a value. For example, the term ${t''}_i^{x \times y} \, \varepsilon_i$ comes from the multiplication of t_l^x by $\alpha_i^y \varepsilon_i$ (and its symmetric), and thus represents the uncertainty on the global first-order error due to the uncertainty on the value at label i. The multiplications $\varepsilon_i \eta_l$ cannot be represented in our linear forms, we then use new noise symbols ϑ_p to model these terms.

Finally, multiplying errors introduce higher-order errors, which are modelled by the following form, where linear correlations are also kept :

$$e_{ho}^x = (t_h^x + \bigoplus_{l \in L_2} {t'}_{h,l}^x \, \eta_l + \bigoplus_{i \in I} {t''}_{h,i}^x \, \varepsilon_i + \bigoplus_{p \in P} \beta_{h,p}^x \, \vartheta_p).$$

Note that this model may be used to propagate initial errors on some variables, and is well suited to sensitivity analysis. Indeed, the coefficients of the model give information of how a given error is propagated in further operations.

A tool (FLUCTUAT) is built on this model. In order to show its usefulness, consider the following program, implementing a simple linear recursive filter of order two, where a new independant input `E` between 0 and 1 is read at each iteration (notation: `E=[0,1.0]`):

```
double S,S0,S1,E,E0,E1; int i;
S=0.0; S0=0.0; E=[0,1.0]; E0=[0,1.0];
for (i=1;i<=...;i++) {
  E1=E0; E0=E; E=[0,1.0]; S1=S0; S0=S;
  S = 0.7*E-E0*1.3+E1*1.1+S0*1.4-S1*0.7; }
```

The FLUCTUAT tool implementing the modelisation of values and errors in three different domains, interval arithmetic, a relational model of values (but not relational on errors [5]) and the model briefly described above gives the following comparative results:

- interval arithmetic based model: `S` is found to be in [-5.6e44,5.6e44], and global error in [-1.8e+31,1.8e+31] (1.93 seconds, 13MB)

- relational analysis on the values only [5]: `S` is found to be in [-1.09,2.76], and global error in [-3.6e+29,3.6e+29] (10 s, 32MB)

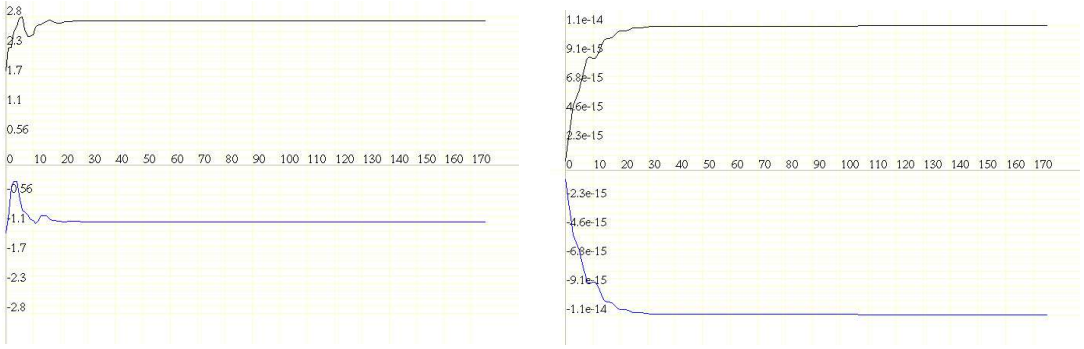
- our relational model on values and errors: `S` is found to be in [-1.09,2.76], and global error in [-1.1e-14,1.1e-14] (5.2 s, 27MB)

Suppose each input has an error in [0,0.001]. Of course, any non-relational model (interval arithmetic for instance) gives an infinite error on the output. With our relational model on values and errors, we still obtain `S` in [-1.09,2.76], but with a stabilized error on the output in [-0.00109,0.00276]. In fact, here the relative error can be shown (in the model) to less or equal to 1: this filter does not amplify the initial errors (sensitivity analysis in the floating-point numbers, and not only in the real numbers).

Finally, we refer to [1] for some examples of industrial codes (mostly control software) we have been able to verify with FLUCTUAT up to now. Current work includes application to scientific codes.

A Complements to the abstract

The evolution of the estimated range of the value and error on `S` in the example we used in the abstract, is shown in Figure 1: these ranges converges in about 100 iterations.



Value in [-1.09,2.76] Error in [-1.1e-14,1.1e-14]

Figure 1:

References

[1] E. Goubault, S. Putot, P. Baufreton and J. Gassino Static Analysis of the Accuracy in Control Systems : Principles and Experiments. Submitted. See http://www.di.ens.fr/~goubault/GOUBAULTpapers.html

[2] J. Stolfi and L. H. de Figueiredo. An introduction to affine arithmetic. TEMA Tend. Mat. Apl. Comput., 4, No. 3 (2003), pp 297-312.

[3] P. Cousot and R. Cousot. Abstract interpretation frameworks. Journal of Logic and Symbolic Computation, 2(4), 1992, pp 511-547.

[4] S. Putot, E. Goubault and M. Martel. Static analysis-based validation of floating-point computations. In LNCS 2991, Springer-Verlag, 2004.

[5] S. Putot, E. Goubault. Static Analysis of Numerical Algorithms In Proceedings of Static Analysis Symposium, 2006, Springer-Verlag.