Robustness Analysis of Finite Precision Implementations

Eric Goubault and Sylvie Putot

CEA Saclay Nano-INNOV, CEA LIST, Laboratory for the Modelling and Analysis of Interacting Systems, Point Courrier 174, 91191 Gif sur Yvette CEDEX
{Eric.Goubault,Sylvie.Putot}@cea.fr

Abstract. A desirable property of control systems is robustness to inputs, when small perturbations of the inputs of a system will cause only small perturbations on outputs. This property should be maintained at the implementation level, where close inputs can lead to different execution paths. The problem becomes crucial for finite precision implementations, where any elementary computation is affected by an error. In this context, almost every test is potentially unstable, that is, for a given input, the finite precision and real numbers paths may differ. Still, state-of-the-art error analyses rely on the stable test hypothesis, yielding unsound error bounds when the conditional block is not robust to uncertainties. We propose a new abstract-interpretation based error analysis of finite precision implementations, which is sound in presence of unstable tests, by bounding the discontinuity error for path divergences. This gives a tractable analysis implemented in the FLUCTUAT analyzer.

1 Introduction

In the analysis of numerical programs, a recurrent difficulty when we want to assess the influence of finite precision on an implementation, is the possibility for a test to be unstable: when, for a given input, the finite precision control flow can differ from the control flow that would be taken by the same execution in real numbers. Not taking this possibility into account may be unsound if the difference of paths leads to a discontinuity in the computation, while taking it into account without special care soon leads to large over-approximations.

This unstable test problem is thus closely related to the notion of continuity/discontinuity in programs, first introduced in [13]. Basically, a program is continuous if, when its inputs are slightly perturbed, its output is also only slightly perturbed, very similarly to the concept of a continuous function. Discontinuity in itself can be a symptom of a major bug in some critical systems, such as the one where a F22 Raptor military aircraft almost crashed after crossing the international date line in 2007, due to a discontinuity in the treatment of dates. We thus want to automatically characterize conditional blocks that perform a continuous treatment of inputs, and are thus robust, and those that do not. Consider the program presented on the left hand side of Figure 1, where input x takes its real value in $[1, 3]$, with an error $0 < u << 1$, that can come from previous finite precision computations or from any uncertainty such as sensor imperfection. The test is potentially unstable: for instance, if the real value of x at control point [1] is $r^x_{[1]} = 2$, then its floating-point value is $f^x_{[1]} = 2 + u$. Thus the execution in real numbers would take the `then` branch and lead at control point [2] to $r^y_{[2]} = r^x_{[1]} + 2 = 4$, whereas the floating-point execution would take the `else` branch and lead to $f^y_{[4]} = f^x_{[1]} = 2 + u$. The test is not only unstable, but also introduces a discontinuity around the test condition ($x == 2$). Indeed, for $r^x_{[1]} = 2$, there is an error due to discontinuity of $f^y_{[4]} - r^y_{[2]} = -2 + u$.

In the rest of the paper, we propose a new analysis, that enhances earlier work by the authors [11], by computing and propagating bounds on those discontinuity errors. This previous work characterized the computation error due to the implementation in finite precision, by comparing the computations in real-numbers with the same computations in the floating-point semantics, relying on the stable test assumption: the floating-point number control flow does not diverge from the real number control flow. When this assumption is not satisfied, the comparison between the two semantics (the error bounds) could be unsound. This issue appears in all other (static or dynamic) existing analyzes of numerical error propagation; the expression unstable test is actually taken from CADNA [4], a stochastic arithmetic instrumentation of programs, to assert their numerical quality. In Hoare provers dealing with both real number and floating-point number semantics, e.g. [1] this issue has to be sorted out by the user, through suitable assertions and lemmas.

Here as in previous work, we rely on the relational abstractions of real number and floating numbers semantics using affine sets (concretized as zonotopes) [9,10,5,6,11]. But we now also, using these abstractions, compute and solve constraints on inputs such that the execution potentially leads to unstable tests, and thus accurately bound the discontinuity errors, computed as the difference of the floating-point value in one branch and the real value in another, when the test distinguishing these two branches can be unstable.

x := [1,3] + u; // [1]
/* $\hat{r}^x_{[1]} = 2 + \varepsilon^r_1$; $\hat{e}^x_{[1]} = u$ */
if (x $\leq$ 2)
y = x+2; // [2]
/* $\hat{r}^y_{[2]} = 4 + \varepsilon^r_1$;
$\hat{e}^y_{[2]} = u + \delta\varepsilon^e_2$ */
else
y = x; // [3]
/* $\hat{r}^y_{[4]} = 2 + \varepsilon^r_1$;
/* $\hat{e}^y_{[4]} = u$ */
// [4] /* $\hat{r}^y_{[4]} = \hat{r}^y_{[2]} \sqcup \hat{r}^y_{[4]}$
$\hat{e}^y_{[4]} = \hat{e}^y_{[2]} \sqcup \hat{e}^y_{[3]} + d^y_{[4]}$ */

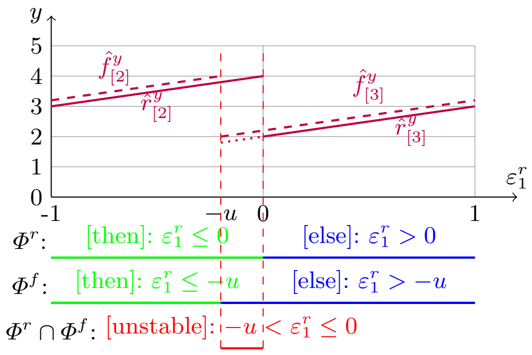

Fig. 1. Running example

Related Work. In [2], the authors introduce a continuity analysis of programs. This approach is pursued in particular in [3], where several refinements of the notion of continuity or robustness of programs are proposed, another one being introduced in [14]. In [14], the algorithm proposed by the authors symbolically traverses program paths and collects constraints on input and output variables. Then for each pair of program paths, the algorithm determines values of input variables that cause the program to follow these two paths and for which the difference in values of the output variable is maximized. One difference between the approaches is that we give extra information concerning the finite precision flow divergence with respect to the real number control flow, potentially exhibiting flawed behaviors. Also, their path-sensitive analysis can exhibit witnesses for worst discontinuity errors, but at the expense of a much bigger combinatorial complexity. Robustness has also been discussed in the context of synthesis and validation of control systems, for instance in [16]. Indeed, robustness has long been central in numerical mathematics, in particular in control theory. The field of robust control is actually concerned in proving stability of controlled systems where parameters are only known in range.

Contents. Our main contribution is a tractable analysis that generalizes both the abstract domain of [11] and the continuity or robustness analyses: it ensures the finite precision error analysis is now sound even in the presence of unstable tests, by computing and propagating discontinuity error bounds for these tests. More details on this analysis and further experiments are available in [12].

2 Preliminaries: Affine Sets for Real Valued Analysis

We sketch here some basics on the abstract domains based on affine sets for the abstraction of real number semantics, necessary to understand the robustness analysis presented here. We refer to [8,9,10,5,6] for more details.

Affine arithmetic is a more accurate extension of interval arithmetic, that takes into account affine correlations between variables. An *affine form* is a formal sum over a set of *noise symbols* ε_i, $\hat{x} \overset{\text{def}}{=} \alpha_0^x + \sum_{i=1}^n \alpha_i^x \varepsilon_i$, with $\alpha_i^x \in \mathbb{R}$ for all i. Each noise symbol ε_i stands for an independent component of the total uncertainty on the quantity $\hat{x}$, its value is unknown but bounded in [-1,1]. The same noise symbol can be shared by several quantities, indicating correlations among them. The result of linear operations on affine forms is an affine form, and is thus interpreted exactly. For non affine operations, an approximate linear resulting form is computed, and bounds for the error committed using this approximate form are used to define a new noise term that makes the resulting form a sound over-approximation.

We use matrix notations to handle affine sets, that is tuples of affine forms. We note $\mathcal{M}(n,p)$ the space of matrices with n lines and p columns of real coefficients. A tuple of affine forms expressing the set of values taken by p variables over n noise symbols ε_i, $1 \leq i \leq n$, is represented by a matrix $A \in \mathcal{M}(n+1,p)$.

Constrained Affine Sets. As described in [6], tests are interpreted by leaving affine sets unchanged and adding some constraints on the ε_i noise symbols, instead of having them vary freely into [-1,1]: we restrain ourselves to executions (or inputs) that can take the considered branch. These constraints can be abstracted in any abstract domain, the simplest being intervals. We note $\mathcal{A}$ for this abstract domain, and use $\gamma : \mathcal{A} \to \wp(\mathbb{R}^n)$ for the concretisation operator, and $\alpha : \wp(\mathbb{R}^n) \to \mathcal{A}$ for some abstraction operator.

This means that abstract values X are now composed of a zonotope identified with its matrix $R^X \in \mathcal{M}(n+1,p)$, together with an abstraction Φ^X of the constraints on the noise symbols, $X = (R^X, \Phi^X)$. The concretisation of such constrained zonotopes or affine sets is $\gamma(X) = \left\{ {}^tC^X \epsilon \mid \epsilon \in \gamma(\Phi^X) \right\}$. For $\Phi^X \in \mathcal{A}$, and $\hat{x}$ an affine form, we note $\Phi^X(\hat{x})$ the interval $[\inf_{\varepsilon \in \gamma(\Phi^X)} \hat{x}(\varepsilon), \sup_{\varepsilon \in \gamma(\Phi^X)} \hat{x}(\varepsilon)]$.

Example 1. On the running example from Figure 1, the real value of input `x`, given in $[1,3]$, will be abstracted by the affine form $\hat{r}^x_{[1]} = 2 + \varepsilon^r_1$, where ε^r_1 is a symbolic variable with values in $[-1,1]$. We associate the abstract value X with $R^X = (2\ 1)$, i.e. $\hat{x} = 2 + \varepsilon_1$, and $\gamma(\Phi^X) = \gamma(\varepsilon_1) = [-1,1]$.

Note the functional abstraction: affine forms represent a function from inputs to variable values. We will use this to interpret tests, and in particular to compute unstable tests conditions. Here, the interpretation of the test `if (x<=2)` in the `then` branch is translated into constraint $2+\varepsilon^r_1 \leq 2$, that is $\varepsilon^r_1 \leq 0$, thus $\gamma(\Phi^X) = [-1,0]$. Then, the interval concretisation of $\hat{x}$ is $\gamma(\hat{x}) = [2-1,2] = [1,2]$.

We also need an upper bound operator to combine abstract values coming from different branches. The computation of upper bounds (and if possible minimal ones) on constrained affine sets is a difficult task, already discussed in several papers [9,10,6,7], and orthogonal to the robustness analysis presented here. We will thus consider we have an upper bound operator on constrained affine sets we note $\sqcup$, and focus on the additional term due to discontinuity in tests.

3 Robustness Analysis of Finite Precision Computations

We now introduce an analysis of finite precision computations, based on an abstraction similar to some previous work [11], but refined to be sound in presence of unstable tests, and to exhibit the potential discontinuity errors due to these tests. For more concision, we insist here on what is directly linked to an accurate treatment of these discontinuities, and rely as much as possible on [11].

Floating-point computation is considered as a perturbation of the same computation in real numbers, and we use zonotopic abstractions of real computations and errors (introducing respectively noise symbols ε^r_i and ε^e_j), from which we get an abstraction of floating point computations. But we make here no assumptions on control flows in tests and will compute branch conditions independently on the real value and the floating-point value. For each branch, we thus get two sets of constraints: $\varepsilon^r = (\varepsilon^r_1, \ldots, \varepsilon^r_n) \in \Phi^X_r$ for the real control flow (test computed on real values R^X), and $(\varepsilon^r, \varepsilon^e) = (\varepsilon^r_1, \ldots, \varepsilon^r_n, \varepsilon^e_1, \ldots, \varepsilon^e_m) \in \Phi^X_f$ for the finite precision control flow (test computed on float values $R^X + E^X$).

Definition 1. *An abstract value X, defined at a given control point, for a program with p variables $x_1, \ldots, x_p$, is thus a tuple $X = (R^X, E^X, D^X, \Phi_r^X, \Phi_f^X)$ composed of the following affine sets and constraints, for all $k = 1, \ldots, p$:*

$$\begin{cases} R^X \ : \ \hat{r}_k^X = r_{0,k}^X + \sum_{i=1}^n r_{i,k}^X \varepsilon_i^r & \text{where } \varepsilon^r \in \Phi_r^X \\ E^X \ : \ \hat{e}_k^X = e_{0,k}^X + \sum_{i=1}^n e_{i,k}^X \varepsilon_i^r + \sum_{j=1}^m e_{n+j,k}^X \varepsilon_j^e & \text{where } (\varepsilon^r, \varepsilon^e) \in \Phi_f^X \\ D^X \ : \ \hat{d}_k^X = d_{0,k}^X + \sum_{i=1}^o d_{i,k}^X \varepsilon_i^d & \\ \qquad \hat{f}_k^X = \hat{r}_k^X + \hat{e}_k^X & \text{where } (\varepsilon^r, \varepsilon^e) \in \Phi_f^X \end{cases}$$

where $R^X \in \mathcal{M}(n+1, p)$ defines the real values of variables, and $\hat{r}_k^X$ giving the real value of x_k is defined on the ε_i^r; $E^X \in \mathcal{M}(n+m+1, p)$ defines the rounding errors (or initial uncertainties) and their propagation through computations using the ε_i^r which handle the uncertainty on the real value and the ε_i^e which handle the uncertainty on the rounding errors; $D^X \in \mathcal{M}(o+1, p)$ defines the discontinuity errors, using noise symbols ε_i^d; Φ_r^X abstracts the set of constraints such that the real control flow reaches the control point, $\varepsilon^r \in \Phi_r^X$, and Φ_f^X abstracts the set of constraints for the finite precision control flow, $(\varepsilon^r, \varepsilon^e) \in \Phi_f^X$.

Example 2. Let us consider the running example. We already saw that the real value of input `x` is abstracted by the affine form $\hat{r}_{[1]}^x = 2 + \varepsilon_1^r$. Its error is $\hat{e}_{[1]}^x = u$ and its finite precision value is $\hat{f}_{[1]}^x = \hat{r}_{[1]}^x + \hat{e}_{[1]}^x = 2 + \varepsilon_1^r + u$.

Test Interpretation. A test `e1 op e2`, where `e1` and `e2` are two arithmetic expressions, and `op` an operator among $\le, <, \ge, >, =, \ne$, is interpreted as `z op 0`, where `z` is the abstraction of expression `e1 - e2` with affine sets. We interpret this test independently for real and floating-point value, relying on the test interpretation on constrained affine sets introduced in [6]:

Definition 2. *Let $X = (R^X, E^X, D^X, \Phi_r^X, \Phi_f^X)$ a constrained affine set. We define $Z = ([\![x_k \ op \ 0]\!]X$ by*

$$\begin{cases} (R^Z, E^Z, D^Z) = (R^X, E^X, D^X) \\ \Phi_r^Z = \Phi_r^X \bigcap \alpha \left(\varepsilon^r \mid r_{0,k}^X + \sum_{i=1}^n r_{i,k}^X \varepsilon_i^r \ op \ 0\right) \\ \Phi_f^Z = \Phi_f^X \bigcap \alpha \left((\varepsilon^r, \varepsilon^e) \mid r_{0,k}^X + e_{0,k}^X + \sum_{i=1}^n (r_{i,k}^X + e_{i,k}^X)\varepsilon_i^r + \sum_{j=1}^m e_{n+j,k}^X \varepsilon_j^e \ op \ 0\right) \end{cases}$$

Example 3. Consider the running example. We start with $\hat{r}_{[1]}^x = 2 + \varepsilon_1^r$, $\hat{e}_{[1]}^x = u$. The condition for the real control flow to take the `then` branch is $\hat{r}_{[1]}^x = 2 + \varepsilon_1^r \le 2$, thus Φ^r is $\varepsilon_1^r \in [-1, 0]$. The condition for the finite precision control flow to take the `then` branch is $\hat{f}_{[1]}^x = \hat{r}_{[1]}^x + \hat{e}_{[1]}^x = 2 + \varepsilon_1^r + u \le 2$, thus Φ^f is $\varepsilon_1^r \in [-1, -u]$. Thus, the unstable test condition being that for the same input the real and float control flow are different, this amounts to intersecting these two conditions on ε_1^r, and yields $-u < \varepsilon_1^r \le 0$. These constraints are illustrated on Figure 1, with $u = 0.2$: Φ_r denotes the constraints on the real value, Φ_f, the constraints on the finite precision value, and $\Phi^r \cap \Phi^f$, the unstable test condition. For the other possibility for an unstable test, that is the execution in real numbers takes the `else` branch while the float execution takes the `then` branch, the constraints are $\varepsilon_1^r < 0$ and $\varepsilon_1^r \le -u$, which are incompatible. This possibility is thus excluded.

Interval Concretisation. The interval concretisation of the value of program variable x_k defined by the abstract value $X = (R^X, E^X, D^X, \Phi_r^X, \Phi_f^X)$, is, with the notations of Section 2:

$$\begin{cases} \gamma_r(\hat{r}_k^X) &= \Phi_r^X(r_{0,k}^X + \sum_{i=1}^n r_{i,k}^X \varepsilon_i^r) \\ \gamma_e(\hat{e}_k^X) &= \Phi_f^X(e_{0,k}^X + \sum_{i=1}^n e_{i,k}^X \varepsilon_i^r + \sum_{j=1}^m e_{n+j,k}^x \varepsilon_j^e) \\ \gamma_d(\hat{d}_k^X) &= \Phi_f^X(d_{0,k}^X + \sum_{l=1}^o d_{l,k}^x \varepsilon_l^d) \\ \gamma_f(\hat{f}_k^X) &= \Phi_f^X(r_{0,k}^X + e_{0,k}^X + \sum_{i=1}^n (r_{i,k}^X + e_{i,k}^X) \varepsilon_i^r + \sum_{j=1}^m e_{n+j,k}^x \varepsilon_j^e) \end{cases}$$

Example 4. Take variable `y` in the running example. In the `then` branch, its real value is $\hat{r}_{[2]}^y = \hat{r}_{[1]}^x + 2 = 4 + \varepsilon_1^r$, the error $\hat{e}_{[2]}^y = \hat{e}_{[1]}^x + \delta\varepsilon_2^e$, where δ is the bound on the elementary rounding error on `y`, due to the addition, we deduce $\hat{f}_{[2]}^y = \hat{r}_{[2]}^y + \hat{e}_{[2]}^y$. In the `else` branch, the real value is $\hat{r}_{[3]}^y = \hat{r}_{[1]}^x = 2 + \varepsilon_1^r$, the error $\hat{e}_{[3]}^y = \hat{e}_{[1]}^x$, and we deduce $\hat{f}_{[3]}^y = \hat{r}_{[3]}^y + \hat{e}_{[3]}^y$. In Figure 1, we represent in solid lines the real value of y and in dashed lines its finite precision value. The interval concretisation of its real value on Φ_r^X, is $\gamma_r(\hat{r}_{[3]}^y) = \Phi_r^X(2 + \varepsilon_1^r) = 2 + [0, 1] = [2, 3]$. The interval concretisation of its floating-point value on Φ_f^X, is $\gamma_f(\hat{f}_{[3]}^y) = \Phi_f^X(\hat{r}_{[3]}^y + u) = 2 + [-u, 1] + u = [2, 3 + u]$. Actually, $\hat{r}_{[3]}^y$ is defined on $\Phi_r^X \cup \Phi_f^X$, as illustrated on Figure 1, because it is both used to abstract the real value, or, perturbed by an error term, to abstract the finite precision value.

Join. If the test distinguishing two branches can be unstable, then when we join abstract values X and Y coming from the two branches, the difference between the floating-point value of X and the real value of Y, $(R^X + E^X) - R^Y$, and the difference between the floating-point value of X and the real value of Y, $(R^Y + E^Y) - R^X$, are also errors due to finite precision. The join of all error terms can then be expressed as $E^Z + D^Z$, where $E^Z = E^X \sqcup E^Y$ is the propagation of classical rounding errors, and D^Z expresses the discontinuity errors.

A key point for an accurate computation of these discontinuity terms, is to express the unstable tests conditions as an intersection of constraints on the ε_i^r noise symbols, yielding a restriction of the sets of inputs (or equivalently the ε_i^r). It is thus crucial that these ε_i^r are shared by affine sets for real and float values.

Definition 3. *We join two abstract values X and Y by $Z = X \sqcup Y$ defined as $Z = (R^Z, E^Z, D^Z, \Phi_r^X \cup \Phi_r^Y, \Phi_f^X \cup \Phi_f^Y)$ where*

$$\begin{cases} (R^Z, \Phi_r^Z \cup \Phi_f^Z) = (R^X, \Phi_r^X \cup \Phi_f^X) \sqcup (R^Y, \Phi_r^Y \cup \Phi_f^Y) \\ (E^Z, \Phi_f^Z) = (E^X, \Phi_f^X) \sqcup (E^Y, \Phi_f^Y) \\ D^Z = D^X \sqcup D^Y \sqcup (R^X - R^Y, \Phi_f^X \sqcap \Phi_r^Y) \sqcup (R^Y - R^X, \Phi_f^Y \sqcap \Phi_r^X) \end{cases}$$

Example 5. Consider variable `y` in the example. We join $X = (\hat{r}_{[2]}^y = 4 + \varepsilon_1^r, \hat{e}_{[2]}^y = u + \delta\varepsilon_2^e, 0, \varepsilon_1^r \in [-1, 0], (\varepsilon_1^r, \varepsilon_2^e) \in [-1, -u] \times [-1, 1])$ from the `then` branch with $Y = (\hat{r}_{[3]}^y = 2 + \varepsilon_1^r, \hat{e}_{[3]}^y = u, 0, \varepsilon_1^r \in [0, 1], \varepsilon_1^r \in [-u, 1])$ from the `else` branch.

With the analysis of [11] that makes the stable test assumption, we get when joining branches at control point [4], $\hat{r}_{[4]}^y = \hat{r}_{[2]}^y \sqcup \hat{r}_{[3]}^y = 3 + \varepsilon_4^r \in [2, 4]$ with new noise symbol ε_4^r (we do not detail here the upper bound operator on affine forms), $\hat{e}^y_{[4]} = \hat{e}^y_{[2]} \sqcup \hat{e}^y_{[3]} = u + \delta\varepsilon_2^e \in [u-\delta, u+\delta]$, and $\hat{f}^y_{[4]} = \hat{r}^y_{[4]} + \hat{e}^y_{[4]} = 3 + u + \varepsilon_4^r + \delta\varepsilon_2^e$. This is sound for the real and float values $\hat{r}^y_{[4]}$ and $\hat{f}^y_{[4]}$, but unsound for the error.

The new analysis also computes bounds for discontinuity errors. The discontinuity due to the first possible unstable test, when the real takes the `then` branch and float takes the `else` branch is: $\hat{r}^y_{[3]} - \hat{r}^y_{[2]} = 2 + \varepsilon_1^r - 4 + \varepsilon_1^r = -2$, for $\varepsilon_1^r \in \Phi_f^Y \cap \Phi_r^X = [-u, 1] \cap [-1, 0] = [-u, 0]$. As already seen, the other possibility of an unstable test is excluded. The error is now $\hat{e}^y_{[4]} + d^y_{[4]}$ where $d^y_{[4]} = -2\chi_{[-u,0]}(\varepsilon_1)$ and $\chi_{[a,b]}(x)$ equals 1 if x is in $[a, b]$ and 0 otherwise.

4 Experiments

We experimented some small examples inspired by industrial codes, using our implementation of this abstraction in our static analyzer FLUCTUAT. More experiments are described in [12].

A Simple Interpolator. The following example implements an interpolator, affine by sub-intervals, as classically found in critical embedded software. It is a robust implementation indeed. In the code below, we used the FLUCTUAT assertion `FREAL_WITH_ERROR(a,b,c,d)` to denote an abstract value (of resulting type `float`), whose corresponding real values are $x \in [a, b]$, and whose corresponding floating-point values are of the form $x + e$, with $e \in [c, d]$.

```
float R1[3], E, res;
R1[0] = 0;  R1[1] = 5 * 2.25; R1[2] = R1[1] + 20 * 1.1;
E = FREAL_WITH_ERROR(0.0,100.0,-0.00001,0.00001);
if (E < 5)
  res = E*2.25 + R1[0];
else if (E < 25)
  res = (E-5)*1.1 + R1[1];
else
  res = R1[2];
return res;
```

The analysis proves `res` in [-2.25e-5,33.2], with an error in [-3.5e-5,2.4e-5], thus of the order of magnitude of the input error, despite unstable tests.

A Simple Square Root Function. This example is a rewrite in some particular case, of an actual implementation of a square root function, in an industrial context:

```
double sqrt2 = 1.414213538169860839843750;
double S, I;  I = DREAL_WITH_ERROR(1,2,0,0.001);
if (I>=2)
  S = sqrt2*(1+(I/2-1)*(.5-0.125*(I/2-1)));
 else
  S = 1+(I-1)*(.5+(I-1)*(-.125+(I-1)*.0625));
```

With the former type of analysis within FLUCTUAT, we get the unsound result that `S` is proven in the real number semantics to be in [1,1.4531] with a global error in [-0.0005312,0.00008592]. The function does not exhibit a big discontinuity, but still larger than these bounds. For I=2 for instance, the `then` branch computes `sqrt2` which is approximately 1.4142, whereas the `else` branch computes 1+0.5-0.125+0.0625=1.4375. With our present analysis, FLUCTUAT proves that S in the real number semantics is in [1,1.4531] with an error in [-0.0394,0.0389], the test discontinuity accounting for most of it ([-0.0389,0.0389], coherent with the above rough estimate of 0.0233).

5 Conclusion

We have proposed an abstract interpretation based static analysis of the robustness of finite precision implementations, as a generalization of both software robustness or continuity analysis and finite precision error analysis, by abstracting the impact of finite precision in numerical computations and control flow divergences. Future work includes going along the lines of [15] and resorting to more sophisticated constraint solving: indeed our analysis can generate constraints on noise symbols, which we only partially use for the time being.

References

1. Boldo, S., Filliâtre, J.-C.: Formal Verification of Floating-Point Programs. In: 18th IEEE International Symposium on Computer Arithmetic (June 2007)
2. Chaudhuri, S., Gulwani, S., Lublinerman, R.: Continuity analysis of programs. In: POPL, pp. 57–70 (2010)
3. Chaudhuri, S., Gulwani, S., Lublinerman, R.: Continuity and robustness of programs. Commun. ACM 55(8), 107–115 (2012)
4. Chesneaux, J.-M., Lamotte, J.-L., Limare, N., Lebars, Y.: On the new cadna library. In: SCAN (2006)
5. Ghorbal, K., Goubault, E., Putot, S.: The zonotope abstract domain taylor1+. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 627–633. Springer, Heidelberg (2009)
6. Ghorbal, K., Goubault, E., Putot, S.: A logical product approach to zonotope intersection. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 212–226. Springer, Heidelberg (2010)
7. Goubault, E., Le Gall, T., Putot, S.: An accurate join for zonotopes, preserving affine input/output relations. In: Proceedings of NSAD 2012. ENTCS, pp. 65–76 (2012)
8. Goubault, E., Putot, S.: Static analysis of numerical algorithms. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 18–34. Springer, Heidelberg (2006)
9. Goubault, E., Putot, S.: Perturbed affine arithmetic for invariant computation in numerical program analysis. CoRR, abs/0807.2961 (2008)
10. Goubault, E., Putot, S.: A zonotopic framework for functional abstractions. CoRR, abs/0910.1763 (2009)
11. Goubault, E., Putot, S.: Static analysis of finite precision computations. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 232–247. Springer, Heidelberg (2011)
12. Goubault, E., Putot, S.: Robustness analysis of finite precision implementationss. CoRR, abs/1309.3910 (2013)
13. Hamlet, D.: Continuity in software systems. In: ISSTA, pp. 196–200 (2002)
14. Majumdar, R., Saha, I.: Symbolic robustness analysis. In: RTSS (2009)
15. Ponsini, O., Michel, C., Rueher, M.: Refining abstract interpretation based value analysis with constraint programming techniques. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 593–607. Springer, Heidelberg (2012)
16. Tabuada, P., Balkan, A., Caliskan, S.Y., Shoukry, Y., Majumdar, R.: Input-output robustness for discrete systems. In: EMSOFT, pp. 217–226 (2012)