Abstract Interpretation of Floating-Point Computations

Sylvie Putot
Laboratory for ModElling and Analysis of Systems in Interaction, CEA-LIST/X/CNRS

Session: Static Analysis for Safety and Performance

ARTIST2 - MOTIVES
Trento - Italy, February 19-23, 2007

Outline

- Introduction
  - Floating-point computations
  - Static analysis and abstract interpretation
- An abstract interpretation for floating-point computations : a relational domain relying on affine arithmetic
  - Introduction to affine arithmetic
  - Relational domain for real value computation
    - arithmetic operations
    - join, meet, order
  - From real to floating-point computation : relational domain for values and errors
- Examples
- References
- Joint work with Eric Goubault

Floating-point numbers (defined by the IEEE 754 norm)

- Normalized floating-point numbers

$$(-1)^s 1.x_1 x_2 \ldots x_n \times 2^e \quad \text{(radix 2 in general)}$$

  - implicit 1 convention ($x_0 = 1$)
  - $n = 23$ for simple precision, $n = 53$ for double precision
  - exponent e is an integer represented on k bits ($k = 8$ for simple precision, $k = 11$ for double precision)

- Denormalized numbers (gradual underflow),

$$(-1)^s 0.x_1 x_2 \ldots x_n \times 2^{e_{\min}}$$

ULP : Unit in the Last Place

- $ulp(x)$ = distance between two consecutive floating-point numbers around x = maximal rounding error of a number around x
- A few figures for simple precision floating-point numbers :

largest normalized	$\sim$	$3.40282347 * 10^{38}$
smallest positive normalized	$\sim$	$1.17549435 * 10^{-38}$
largest positive denormalized	$\sim$	$1.17549421 * 10^{-38}$
smallest positive denormalized	$\sim$	$1.40129846 * 10^{-45}$
$ulp(1)$	$=$	$2^{-23} \sim 1.19200928955 * 10^{-7}$

Some difficulties of floating-point computation

- Representation error : transcendental numbers π, e, but also

$$\frac{1}{10} = 0.000110011001100110011001100 \cdots$$

- Floating-point arithmetic :
  - absorption : $1 + 10^{-8} = 1$ in simple precision float
  - associative law not true : $(-1+1)+10^{-8} \neq -1+(1+10^{-8})$
  - cancellation : important loss of relative precision when two close numbers are subtracted
- Some more trouble :
  - re-ordering of operations by the compiler
  - storage of intermediate computation either in register or in memory, with different floating-point formats

Example of cancellation : surface of a flat triangle

(a, b, c the lengths of the sides of the triangle, a close to $b + c$):

$$A = \sqrt{s(s-a)(s-b)(s-c)} \qquad s = \frac{a+b+c}{2}$$

Then if a,b, or c is known with some imprecision, $s - a$ is very inaccurate. Example,

real number	floating-point number
$a = 1.9999999$	$a = 1.999999881...$
$b = c = 1$	$b = c = 1$
$s - a = 5e - 08$	$s - a = 1.19209e - 07$
$A = 3.16...e - 4$	$A = 4.88...e - 4$

In real world : a catastrophic example

- 25/02/91: a Patriot missile misses a Scud in Dharan and crashes on an american building : 28 deads.
- Cause :
  - the missile program had been running for 100 hours, incrementing an integer every 0.1 second
  - but 0.1 not representable in a finite number of digits in base 2

$$\frac{1}{10} = 0.0001100110011001100110 0\cdots$$

Truncation error	$\sim$	0.000000095 (decimal)
Drift, on 100 hours	$\sim$	$0.34s$
Location error on the scud	$\sim$	$500m$

But also some other costly errors ...

- Explosion of Ariane 5 in 1996 (conversion of a 64 bits float into a 16 bits integer : overflow)
- Vancouver stock exchange in 1982
  - index introduced with initial value 1000.000
  - after each transaction, updated and truncated to the 3rd fractional digit
  - within a few months : index=524.881, correct value 1098.811
  - explanation : biais. The errors all have same sign
- Sinking of an offshore oil platform in 1992 : inaccurate finite element approximation

Collection of Software Bugs at url
`http://www5.in.tum.de/~huckle/bugse.html`

Validation of accuracy "by hand" ?

- A popular way : try the algorithm with different precision (using matlab for example) and compare the results
- Example (by Rump) : in FORTRAN on an IBM S/370, computing with $x = 77617$ and $y = 33096$ and $x_1 = \frac{61.0}{11}$,

  $$f = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

  gives :
  - in single precision, $f = 1.172603...$
  - in double precision, $f = 1.1726039400531...$
  - in extended precision, $f = 1.172603940053178...$
- We would deduce computation is correct ?
- True value is $f = -0.82739...$!!!

IEEE 754 norm : correct (or exact) rounding

- The user chooses one among four rounding modes (rounding to the nearest which is the default mode, rounding towards $+\infty$, rounding towards $-\infty$, or rounding towards 0)
- The result of $x * y$, $*$ being $+, -, \times, /$ or of $\sqrt{x}$, is the rounded value of the real result (thus the rounding error is less than the ulp of the result)

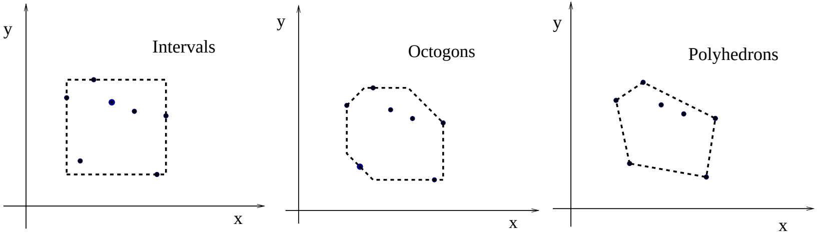
$\rightarrow$ Allows to prove some properties on programs using floating-point numbers

Static Analysis

- Analysis of the source source, for a set of inputs and parameters, without executing it
- The program is considered as a discrete dynamical system
- Find in an automatic, and guaranteed way :
  - invariant properties (true on all trajectories - for all possible inputs or parameters).
    Example : bounds on values of variables
  - liveness properties (that become true at some moment on one trajectory).
    Examples : state reachability, termination

But undecidable in general

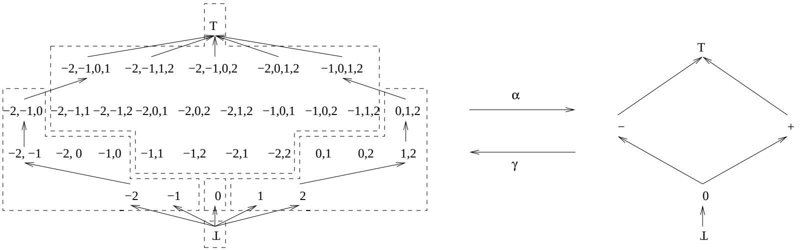
Thus abstraction to compute over-approximations of sets of values : Abstract Interpretation



The analysis must terminate, may return an over-approximated information ("false alarm"), but never a false answer

Abstract Interpretation (Cousot & Cousot 77)

Theory of semantics approximation (operators, fixpoint transfers)



Fixpoint computation

To automatically find local invariants :

- Abstract domain (lattice) for sets of value
- The semantic is given by a system of equations, of which we compute iteratively a fixpoint :

$$X = \begin{pmatrix} X_1 \\ \dots \\ X_n \end{pmatrix} = F \begin{pmatrix} X_1 \\ \dots \\ X_n \end{pmatrix}$$

- F is non-decreasing, least fixpoint is the limit of Kleene iteration $X^0 = \perp$, $X^1 = F(X^0)$, $\dots$, $X^{k+1} = X^k \cup F(X^k)$, $\dots$
- Iteration strategies, extrapolation (called widenings) to reach a fixpoint in finite time

Example : lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\perp$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$
- Sup : $[a, b] \cup [c, d] = [\min(a, b), \max(c, d)]$
- Inf : $[a, b] \cap [c, d] = [\max(a, b), \min(c, d)]$

Example

```
int x=0;               // 1
while (x<100) {        // 2
  x=x+1;               // 3
}                      // 4
```

$$
\begin{aligned}
x_1 &= [0,0] \\
x_2 &=]-\infty, 99] \cap (x_1 \cup x_3) \\
x_3 &= x_2 + [1,1] \\
x_4 &= [100, +\infty[\cap (x_1 \cup x_3)
\end{aligned}
$$

- Iterate $i+1$ $(i < 100)$ [Kleene/Jacobi/Gauss-Seidl] :

$$
\begin{aligned}
x_2^1 &= [0,0] & \quad x_2^{i+1} &= [0,i] \\
x_3^1 &= [1,1] & \quad x_3^{i+1} &= [1,i+1] \\
x_4^1 &= \perp & \quad x_4^{i+1} &= \perp
\end{aligned}
$$

- Fixpoint (after 101 Kleene iterates or widening) :

$$x_2^\infty = [0,99]; \quad x_3^\infty = [1,100]; \quad x_4^\infty = [100,100]$$

Analysis of programs using floating-point numbers

What is a correct program when using floating-point numbers ?

- No run-time error, such as division by 0
- But also the program does compute what is expected with respect to some tolerance (the programmer usually thinks in real numbers)

For that, we need :

- Bounds of floating-point values (ASTREE, FLUCTUAT)
- Bounds on the discrepancy error between the real and floating-point computations (FLUCTUAT)
- If possible, the main source of this error (FLUCTUAT)

Related work and tools

- The ASTREE static analyzer (see references)
  - Detection of run-time error for large synchronous instrumentation software
  - Using in particular octogons and domains specialized for order 2 filters (ellipsoids)
  - Taking floating-point arithmetic into account

  `http://www.astree.ens.fr/`
- CADNA : estimation of the roundoff propagation in scientific programs by stochastic testing
  `http://www-anp.lip6.fr/cadna/`
- GAPPA : automatic proof generation of arithmetic properties
  `http://lipforge.ens-lyon.fr/www/gappa/`

Analysis for the floating-point value

- First natural idea : Interval Arithmetic (IA) with floating-point bounds, where min bound computed with rounding to $-\infty$ and max bound computed with rounding to $+\infty$
  - $[a, b] + [c, d] = [a + c, b + d]$
  - $[a, b] - [c, d] = [a - d, b - c]$
  - $[a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- Defect : too conservative, non relational
  - extreme example : if $X = [-1, 1]$, $X - X$ computed in interval arithmetic is not 0 but $[-2, 2]$
- A solution : Affine Arithmetic, an extension of IA that takes linear correlations into account
  - but correlations true only for computations on real numbers

Affine Arithmetic for real numbers

- Proposed in 1993 by Comba, de Figueiredo and Stolfi as a more accurate extension of Interval Arithmetic
- A variable x is represented by an affine form $\hat{x}$:

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n,$$

  where $x_i \in \mathbb{R}$ and ε_i are independent symbolic variables with unknown value in $[-1, 1]$.
  - $x_0 \in \mathbb{R}$ is the *central value* of the affine form
  - the coefficients $x_i \in \mathbb{R}$ are the *partial deviations*
  - the ε_i are the *noise symbols*
- The sharing of noise symbols between variables expresses *implicit dependency*

Affine arithmetic : arithmetic operations

- *Assignment* of a of a variable x whose value is given in a range $[a, b]$ introduces a noise symbol ε_i :

$$\hat{x} = \frac{(a+b)}{2} + \frac{(b-a)}{2}\,\varepsilon_i.$$

- *Addition* is computed componentwise (no new noise symbol):

$$\hat{x} + \hat{y} = (\alpha_0^x + \alpha_0^y) + (\alpha_1^x + \alpha_1^y)\varepsilon_1 + \ldots + (\alpha_n^x + \alpha_n^y)\varepsilon_n$$

For example, with real (exact) coefficients , $f - f = 0$.

- *Multiplication* : we select an approximate linear form, the approximation error creates a new noise term :

$$\hat{x} \times \hat{y} = \alpha_0^x \alpha_0^y + \sum_{i=1}^{n} (\alpha_i^x \alpha_0^y + \alpha_i^y \alpha_0^x)\varepsilon_i + (\sum_{i=1}^{n} |\alpha_i^x| . | \sum_{i=1}^{n} |\alpha_i^y|)\varepsilon_{n+1}.$$

Affine forms define implicit relations : example

Consider, with $a \in [-1, 1]$ and $b \in [-1, 1]$, the expressions

```
x = 1 + a + 2 * b;
y = 2 - a;
z = x + y - 2 * b;
```

- The representation as affine forms is $\hat{x} = 1 + \epsilon_1 + 2\epsilon_2$, $\hat{y} = 2 - \epsilon_1$, with noise symbols $\epsilon_1, \epsilon_2 \in [-1, 1]$
- This implies $x \in [-2, 4]$, $y \in [1, 3]$
- It also contains implicit relations, such as $x + y = 3 + 2\epsilon_2 \in [1, 5]$ or $x + y - 2b = 3$: we thus get
$$\boxed{z = x + y - 2b = 3}$$
- Whereas we get with intervals
$$\boxed{z = x + y - 2b \in [-3, 9]}$$

Affine forms and existing relational domains

- *More expressive* (less abstract) than zones or octogons [A. Mine]
- Close to *dynamic templates* [Z. Manna]
- Provides *Sub-polyedric* relations (there is *a concretization* to center-symmetric bounded convex polyedra)
- But by *some aspects* better than polyhedra [P. Cousot/N. Halbwachs]
  - for example, to interpret *non-linear computations* :
    - dynamic linearization of non-linear computations
  - much more efficient in *computation time and memory*
    - dynamic construction of relations
    - no static packing of variables needed

Comparative example

x = [0,2]
y = x+[0,2]

z = xy;
t = z-2*x-y;

Zones/polyhedra (with a simple semantics):

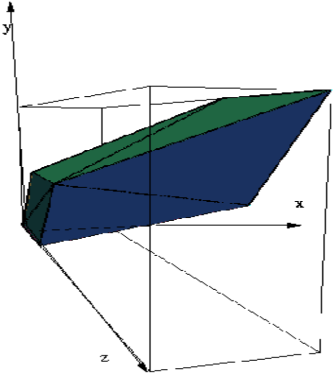
$$\begin{cases} 0 \leq x \leq 2 \\ 0 \leq y - x \leq 2 \\ 0 \leq z \leq 8 \\ -8 \leq t \leq 8 \end{cases}$$

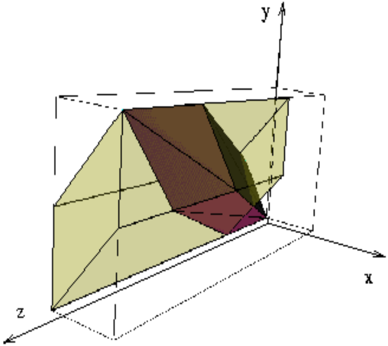
Affine forms:

$$\begin{cases} x = 1 + \varepsilon_1 & \in [0, 2] \\ y = 2 + \varepsilon_1 + \varepsilon_2 & \in [0, 4] \\ z = 2.5 + 3\,\varepsilon_1 + \varepsilon_2 + 1.5\,\varepsilon_3 & \in [-3, 8] \\ t = -1.5 + 1.5\,\varepsilon_3 & \in [-3, 0] \end{cases}$$

(in practice coupled with intervals, thus $z \in [0, 8]$)

Concretisation of affine forms (x,y,z)

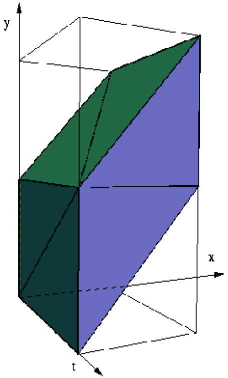


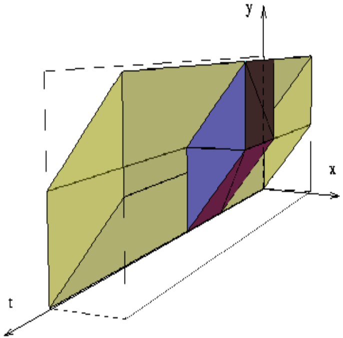


concretization of affine form finds $z - 2x - y \in [-3, 0]$

with classical polyhedron

Concretisation of affine forms (x,y,t)





concretization of affine form with classical polyhedron

Implementation using floating-point numbers

- For the computation of the affine form for the real value, the analyzer also uses finite precision arithmetic :
  - Affine form with floating point coefficients (with higher precision floating-point numbers, using the MPFR library)
  - Uncertainty in the computation of coefficients is handled by creating new noise terms

Join (and meet) operations on affine forms

- Let $[\alpha_i^x \cup \alpha_i^y] = [\alpha_i^x, \alpha_i^y]$ if $\alpha_i^x \leq \alpha_i^y$ else $[\alpha_i^y, \alpha_i^x]$
- A natural join between $\hat{x}$ and $\hat{y}$ is

$$\hat{x} \cup \hat{y} = [\alpha_0^x \cup \alpha_0^y] + \sum_{i \in L} [\alpha_i^x \cup \alpha_i^y] \, \varepsilon_i$$

  Result might be greater than the union of enclosing intervals, but may be more interesting to keep correlations

- But with interval coefficients $\boxed{(\hat{x} \cup \hat{y}) - (\hat{x} \cup \hat{y}) \neq 0}$ we get back to the defects of intervals

Join (and meet) operations on affine forms

For an interval $\mathbf{i}$, we note

$$\text{mid}(\mathbf{i}) = \frac{\underline{i} + \overline{i}}{2}, \quad \text{dev}(\mathbf{i}) = \overline{i} - \text{mid}(\mathbf{i})$$

the center and deviation of the interval.

- A better join is then

$$\hat{x} \cup \hat{y} = \text{mid}([\alpha_0^x, \alpha_0^y]) + \sum_{i \in L} \text{mid}([\alpha_i^x, \alpha_i^y])\, \varepsilon_i + \sum_{i \in L \cup \{0\}} \text{dev}([\alpha_i^x, \alpha_i^y])\, \varepsilon_k^u$$

- Then we have affine forms with real coefficients again
- Order on affine forms considers noise symbols due to join operations differently than noise symbols due to arithmetic operations

Example (join)

Let $\hat{x} = 1 + 2\varepsilon_1 + \varepsilon_2$ and $\hat{y} = 2 - \varepsilon_1$.

- Join on intervals : $[x] \cup [y] \in [-2, 4]$
- First join on affine forms :
  - $\hat{x} \cup \hat{y} = [1, 2] + [-1, 2]\varepsilon_1 + [0, 1]\varepsilon_2 \subset [-2, 5]$
  - larger enclosure than on intervals but it may still be interesting for further computations to keep relations
- Second join on affine forms :
  - $\hat{x} \cup \hat{y} = 1.5 + 0.5\varepsilon_1 + 0.5\varepsilon_2 + 2.5\varepsilon_3^u \subset [-2, 5]$
  - same enclosure, but $(\hat{x} \cup \hat{y}) - (\hat{x} \cup \hat{y}) = 0$

Order on affine forms with real coefficients

- For variable x, let $\alpha_i^x, i \in L$ denote terms due to "classical" noise symbols and β_k^x denote terms due to "union" noise symbols :

$$\hat{x} \leq \hat{y} \text{ iff } \sum_{i \in L \cup \{0\}} |\alpha_i^x - \alpha_i^y| \leq \sum_k |\beta_k^y| - \sum_k |\beta_k^x|$$

- Projection of "union" noise symbols on "classical" noise symbols in arithmetic operations
- Then we have a complete partial order (under some restrictions)

Correctness of the semantics on affine forms

- Affine forms define *implicit* relations
  - the concretization of an affine form representing a variable must contain the concrete values of the variable
  - and in whatever expression using the affine forms, the concretization as interval of the expression must contain the concrete values it can take
    - we must not introduce non-existing relations by undue sharing of noise symbols

From real to floating-point computation

- Affine arithmetic uses symbolic properties of real number computation, such as associativity and distributivity of $+, \times$
- These properties do not hold exactly for floating-point numbers, thus affine arithmetic can not be directly used for floating-point estimation
- Example :
  - let $x \in [0,2]$ and $y \in [0,2]$, we consider $((x+y)-x)-y$.
  - with affine arithmetic : $x = 1+\varepsilon_1$, $y = 1+\varepsilon_2$
    $((x+y)-x)-y = ((2+\varepsilon_1+\varepsilon_2)-1-\varepsilon_1)-1-\varepsilon_2 = 0$
  - false in floating-point numbers : take $x = 2$ and $y = 0.1$, then in simple precision $((x+y)-x)-y = -9.685755e-08$

Overview for floating-point computation

- Affine arithmetic for real number estimation
- Estimation of the loss of precision due to the use of floating-point numbers
  - using ideas from affine arithmetic
  - decomposition of errors on their provenance in the program
- We deduce bounds for the floating-point value

Representation of values (concrete)

The set of floating-point values that a variable x can take is expressed as:

$$\begin{aligned} f^x &= r^x + e_1^x + e_{ho}^x \\ &= r^x + \bigoplus_{i \in I} \alpha_i^x + e_{ho}^x \end{aligned}$$

where:

- r^x is the real-number value that would have been computed if we had exact arithmetic available
- α_i^x is the coefficient expressing the first-order error introduced by the arithmetic operation labelled i in the program, propagated on x
- e_{ho}^x is the higher-order error

Example

```
float x = 0.1; // [1]
float y = 0.5; // [2]
float z = x+y; // [3]
float t = x*y; // [4]
```

$$
\begin{aligned}
x &= 0.1 + 1.49011612e^{-9}\ [1] \\
y &= 0.5 \\
z &= 0.6 + 1.49011612e^{-9}\ [1] + \\
&\quad 2.23517418e^{-8}\ [3] \\
t &= 0.06 + 1.04308132e^{-9}\ [1] \\
&\quad +2.23517422e^{-9}\ [3] \\
&\quad -8.94069707e^{-10}\ [4] \\
&\quad -3.55271366e^{-17}\ [ho]
\end{aligned}
$$

Abstraction

- Affine Arithmetic for the real part r^x as already presented
- First natural idea: use interval arithmetic for coefficients α_i^x and e_{ho}^x
- Rounding errors given by the IEEE 754 standard:
  - in general, an interval of width $\mathtt{ulp}(x)$ when x is not just a singleton
- But of course, we run into dependency problems : affine arithmetic on errors also

First-order errors

Also represented in affine arithmetic (with other noise symbols):

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

- t_l^x: center of the first-order error associated to the operation l
- ${t'}_l^x \, \eta_l$: deviation on the first-order error associated to the operation l
- the other terms are useful for modelling the propagation of the first-order error terms after non-linear operations
  - For instance, the term ${t''}_i^{x \times y} \, \varepsilon_i$ comes from the multiplication of t_l^x by $\alpha_i^y \varepsilon_i$, and represents the uncertainty on the 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 use *a* new noise symbol ϑ_p

Example : a non linear Newton scheme

Computes the inverse of A, that can take any value in [20,30] :

```
double xi, xsi, A, temp;
signed int *PtrA, *Ptrxi, cond, exp, i;

A = __BUILTIN_DAED_DBETWEEN(20.0,30.0);

/* initial condition = inverse of nearest power of 2 */
PtrA = (signed int *) (&A);
Ptrxi = (signed int *) (&xi);
exp = (signed int) ((PtrA[0] & 0x7FF00000) >> 20) - 1023;
xi = 1; Ptrxi[0] = ((1023-exp) << 20);

temp = xsi-xi; i = 0;
while (abs(temp) > e-10) {
  xsi = 2*xi-A*xi*xi;
  temp = xsi-xi;
  xi = xsi;
  i++;
}
```

Analysis of the inverse computation

- Symbolic execution
  - A = 20.0 : i = 5, xi = 5.0e-2 + [-2.82e-18,-2.76e-18]
  - A = 30.0 : i = 9, xi = 3.33e-2 + [-5.28e-18,6.21e-18]
- Static analysis for A in [20.0,30.0] :
  - Non relational : analysis *does not prove termination* of the Newton algorithm
  - Relational (with 10000 subdivisions) : analysis finds

    $\boxed{\text{i in [5,9], xi in [3.33e-2,5.0e-2]+ [-4.21e-13,4.21e-13]}}$
- Study of this algorithm is not obvious (for example, execution of the same algorithm but with simple precision float variables does not always terminate)

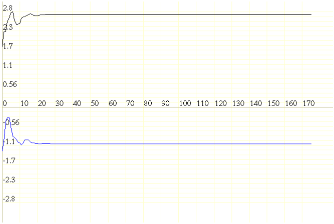
Example : second-order filter

A new independent input E at each iteration of the filter:

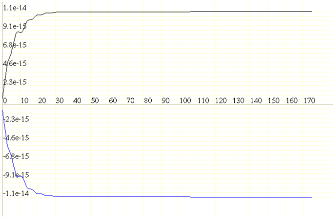
```
double S,S0,S1,E,E0,E1;
int i;

S=0.0; S0=0.0;
E=__BUILTIN_DAED_DBETWEEN(0,1.0);
E0=__BUILTIN_DAED_DBETWEEN(0,1.0);

for (i=1;i<=170;i++) {
  E1 = E0;
  E0 = E;
  E = __BUILTIN_DAED_DBETWEEN(0,1.0);
  S1 = S0;
  S0 = S;
  S = 0.7 * E - E0 * 1.3 + E1 * 1.1 + S0 * 1.4 - S1 * 0.7 ;
}
```

Second-order filter

Relational analysis on values and errors :



Values in [-1.09,2.76]

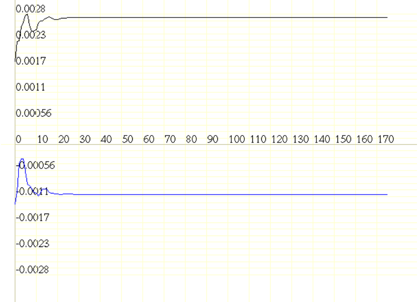


Error in [-1.1e-14,1.1e-14]

Second-order filter

Propagation of an error on the input:

- Each input has now an error in [0,0.001]
- Relational on errors : `S` in [-1.09,2.76], with a stabilized error in [-0.00109,0.00276]



References

- What Every Computer Scientist Should Know About Floating-Point Arithmetic, by D. Goldberg, ACM Computing Surveys, 1991

http://cch.loria.fr/documentation/IEEE754/ACM/goldberg.pdf

- An Introduction to Affine Arithmetic, by J. Stolfi and L.H. de Figueiredo, TEMA 2003

http://www.sbmac.org.br/tema/seletas/docs/v4_3/101_01summary.pdf

- Abstract Interpretation: Achievements and Perspectives, by P. Cousot, SSGRR 2000

http://www.di.ens.fr/~cousot/COUSOTpapers/SSGRRP-00-PC.shtml

References

- A static analyzer for large safety-critical software, by B. Blanchet, P. and R. Cousot, J. Feret, L. Mauborgne, A. Min, D. Monniaux and X. Rival, PLDI 2003

`http://www.di.ens.fr/~cousot/COUSOTpapers/PLDI03.shtml`

- Static analysis-based validation of floating-point computations, by S. Putot, E. Goubault and M. Martel, Dagstuhl Seminar, LNCS 2991, Springer-Verlag, 2004.

`http://www.di.ens.fr/~goubault/papers/SPutot_DagstuhlFinal.ps.gz`

- Static Analysis of Numerical Algorithms, by E. Goubault and S. Putot, SAS 2006

`http://www-ist.cea.fr/publicea/exl-php/200600004467-static-analysis-of-numerical-algorithms.html`