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
  \((-1)^s 1.x_1 x_2 \ldots x_n \times 2^e\) (radix 2 in general)
  - implicit 1 convention \((x_0 = 1)\)
  - \(n = 23\) for simple precision, \(n = 52\) 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),
  \((-1)^s 0.x_1 x_2 \ldots x_n \times 2^{e_{\text{min}}}\)
ULP : Unit in the Last Place

- $\text{ulp}(x) = \text{distance between two consecutive floating-point numbers around } x = \text{maximal rounding error of a number around } x$

- A few figures for simple precision floating-point numbers:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>largest normalized</td>
<td>$3.40282347 \times 10^{38}$</td>
</tr>
<tr>
<td>smallest positive normalized</td>
<td>$1.17549435 \times 10^{-38}$</td>
</tr>
<tr>
<td>largest positive denormalized</td>
<td>$1.17549421 \times 10^{-38}$</td>
</tr>
<tr>
<td>smallest positive denormalized</td>
<td>$1.40129846 \times 10^{-45}$</td>
</tr>
<tr>
<td>$\text{ulp}(1)$</td>
<td>$2^{-23} \approx 1.19200928955 \times 10^{-7}$</td>
</tr>
</tbody>
</table>
Some difficulties of floating-point computation

- Representation error: transcendental numbers $\pi$, $e$, but also
  \[
  \frac{1}{10} = 0.0001100110011001100110011001100 \cdots
  \]

- 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 for analysis:
  - 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):

\[ A = \sqrt{s(s-a)(s-b)(s-c)} \quad s = \frac{a+b+c}{2} \]

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

<table>
<thead>
<tr>
<th>real number</th>
<th>floating-point number</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 1.9999999</td>
<td>a = 1.999999881…</td>
</tr>
<tr>
<td>b = c = 1</td>
<td>b = c = 1</td>
</tr>
<tr>
<td>s – a = 5e – 08</td>
<td>s – a = 1.19209e – 07</td>
</tr>
<tr>
<td>A = 3.16…e – 4</td>
<td>A = 4.88…e – 4</td>
</tr>
</tbody>
</table>
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 is not representable in a finite number of digits in base 2
    \[
    \frac{1}{10} = 0.00011001100110011001100\ldots
    \]
    - Truncation error \( \approx 0.000000095 \) (decimal)
    - Drift, on 100 hours \( \approx 0.34 \) s
    - Location error on the scud \( \approx 500 \) m
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 $y = 33096$,

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

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 $x \times y$, ($\times$ being $+,-,\times,/\$), or of $\sqrt{x}$, is the rounded value (with chosen rounding mode) of the real result
- thus the rounding error of such an operation is always less than the ulp of the result

→ This specification 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
- 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)

Sylvie Putot  MEASI, CEA-LIST/X/CNRS  
Abstract Interpretation of Floating-Point Computations
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:

\[
X = \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} = F \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix}
\]

- \( F \) is non-decreasing, least fixpoint is the limit of Kleene iteration \( X^0 = \perp, X^1 = F(X^0), \ldots, X^{k+1} = X^k \cup F(X^k), \ldots \)
- 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 $\bot$ 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: $[a, b] \cup [c, d] = [\min(a, c), \max(b, d)]$
- Inf: $[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$
Example

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

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

  $$
  x_1^2 = [0, 0] \quad x_2^2 = [0, 1] \quad x_3^2 = [1, 2] \quad x_4^2 = \perp
  $$

- **Fixpoint** (after 101 Kleene iterates or widening/narrowing) :

  $$
  x_1^\infty = [0, 99]; \quad x_2^\infty = [1, 100]; \quad x_3^\infty = [100, 100]
  $$
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, overflow, etc
- But also the program does compute something “not too far” from what is expected (= the result of the computation in real numbers, as 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 octagons 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/

- GAPP A : 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, computed with the rounding mode chosen by the user for execution of his program
  - \([a, b] + [c, d] = [a + c, b + d]\)
  - \([a, b] - [c, d] = [a - d, b - c]\)
  - \([a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]\)

- 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 the $\varepsilon_i$ 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 $\varepsilon_i$ are the *noise symbols*

The sharing of noise symbols between variables expresses *implicit dependency*
Concretization as a center-symmetric convex polytope

Concretization \((x, y)\) for the two affine forms

\[
\begin{align*}
    x &= 20 - 4\varepsilon_1 + 2\varepsilon_3 + 3\varepsilon_4 \\
    y &= 10 - 2\varepsilon_1 + \varepsilon_2 - \varepsilon_4
\end{align*}
\]
Affine arithmetic: arithmetic operations

- **Assignment** of a variable $x$ whose value is given in a range $[a, b]$ introduces a noise symbol $\varepsilon_i$:
  \[
  \hat{x} = \frac{(a + b)}{2} + \frac{(b - a)}{2} \varepsilon_i.
  \]

- **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, $\hat{x} - \hat{x} = 0$.

- **Multiplication**: we select an approximate linear form, the approximation error creates a new noise term:
  \[
  \hat{x} \times \hat{y} = \alpha_0^x \alpha_0^y + \sum_{i=1}^{n} (\alpha_i^x \alpha_0^y + \alpha_i^y \alpha_0^x)\varepsilon_i + \left(\sum_{i=1}^{n} |\alpha_i^x| \cdot |\sum_{i=1}^{n} \alpha_i^y|\right)\varepsilon_{n+1}.
  \]
Affine forms define implicit relations: example

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

\[
\begin{align*}
x &= 1 + a + 2 \times b; \\
y &= 2 - a; \\
z &= x + y - 2 \times b;
\end{align*}
\]

- 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 \( \hat{x} \in [-2, 4], \hat{y} \in [1, 3] \) (same as I.A.)
- It also contains implicit relations, such as \( \hat{x} + \hat{y} = 3 + 2 \epsilon_2 \in [1, 5] \) or \( \hat{z} = \hat{x} + \hat{y} - 2b = 3 \)
- Whereas we get with intervals

\[
z = x + y - 2b \in [-3, 9]
\]
Affine forms and existing relational domains

- More expressive than octagons ($\pm x \pm y \leq c$) [A. Mine]
- Provides Sub-polyhedral relations (there is a concretization to center-symmetric bounded convex polytope)
- 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
- Close to dynamic templates [Z. Manna]
Comparative example

\[ x = [0,2] \]
\[ y = x+[0,2] \]
\[ z = xy; \]
\[ t = z-2*x-y; \]

Zones/polyhedra (with a simple semantics):

\[
\begin{aligned}
0 & \leq x \leq 2 \\
0 & \leq y - x \leq 2 \\
0 & \leq z \leq 8 \\
-8 & \leq t \leq 8 \\
\end{aligned}
\]

Affine forms:

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

(in practice coupled with intervals, thus \( z \in [0, 8] \))
Concretisation of affine forms ($x, y, z$)

The concretization of the 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

Sylvie Putot  MEASI, CEA-LIST/X/CNRS

Abstract Interpretation of Floating-Point Computations
Implementation using floating-point numbers

- For implementation of affine forms, we do not have real but floating-point coefficients (possibly higher precision fp numbers using MPFR library)
- One solution is to compute each coefficient of the affine form with intervals of f.p. numbers with outward rounding
  - inaccurate because of intervals
- More accurate: keep point coefficients and handle uncertainty on these coefficients by creating new noise terms
Join operation on affine forms

- Let \([ \alpha^x_i \cup \alpha^y_i ] = [ \alpha^x_i, \alpha^y_i ]\) if \(\alpha^x_i \leq \alpha^y_i\) else \([ \alpha^y_i, \alpha^x_i ]\)

- A natural join between \(\hat{x}\) and \(\hat{y}\) is

\[
\hat{x} \cup \hat{y} = [ \alpha^x_0 \cup \alpha^y_0 ] + \sum_{i \in L} [ \alpha^x_i \cup \alpha^y_i ] \varepsilon_i
\]

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

- But with interval coefficients \((\hat{x} \cup \hat{y}) - (\hat{x} \cup \hat{y}) \neq 0\) we get back to the defects of intervals
Join operation on affine forms

For an interval $i$, we note

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

the center and deviation of the interval.

- A better join is then

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

- 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 :
  - $\hat{x} \cup \hat{y} = 1.5 + 0.5\varepsilon_1 + 0.5\varepsilon_2 + 2.5\varepsilon_3 \subset [-2, 5]$
  - same enclosure, but $(\hat{x} \cup \hat{y}) - (\hat{x} \cup \hat{y}) = 0$
For variable $x$, let $\alpha^x_i, i \in L$ denote terms due to “classical” noise symbols and $\beta^x_k$ denote terms due to “union” noise symbols:

$$\hat{x} \leq \hat{y} \iff \sum_{i \in L \cup \{0\}} |\alpha^x_i - \alpha^y_i| \leq \sum_k |\beta^y_k| - \sum_k |\beta^x_k|$$

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
    - for example we must not introduce non-existing relations by undue sharing of noise symbols
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
  - decomposition of errors on their provenance in the program
  - using noise terms to represent elementary rounding errors
- We deduce bounds for the floating-point value
  - by the sum of real values and errors
  - combined with errors on bounds, that can be in some cases computed much more accurately than maximum error
The set of floating-point values that a variable $x$ can take is expressed as:

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

where:

- $r^x$ is the real-number value that would have been computed if we had exact arithmetic available
- $\alpha_i^x$ 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

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

\[
x = 0.1 + 1.49e^{-9} \quad [1] \\
y = 0.5 \\
z = 0.6 + 1.49e^{-9} \quad [1] + 2.23e^{-8} \quad [3] \\
t = 0.06 + 1.04e^{-9} \quad [1] + 2.23e^{-9} \quad [3] - 8.94e^{-10} \quad [4] - 3.55e^{-17} \quad [ho]
\]
Affine Arithmetic for the real part $r_x$ as already presented

Rounding errors given by the IEEE 754 standard:

- an interval of width $\text{ulp}(x)$ when $x$ is not just a singleton
- if represented with interval arithmetic, we run into dependency problems
- then use affine arithmetic: each of these errors is a new noise term
First-order errors

\[ e_1^x = \bigoplus_{l \in L} t_i^x + \bigoplus_{l \in L} t'_i^x \eta_l \]

- \( t_i^x \): center of the first-order error associated to the operation \( l \)
- \( t'_i^x \eta_l \): deviation on the first-order error associated to operation \( l \)
First-order errors

\[ e_1^x = \bigoplus_{l \in L} t_i^x + \bigoplus_{l \in L} t'_i^x \eta_i + \]

- \( t_i^x \): center of the first-order error associated to the operation \( l \)
- \( t'_i^x \eta_i \): deviation on the first-order error associated to operation \( l \)
- the other terms are useful for modelling the propagation of the first-order error terms after non-linear operations
First-order errors

\[ e_1^x = \bigoplus_{l \in L} t_l^x + \bigoplus_{l \in L} t'_l^x \eta_l + \bigoplus_{i \in I} t''_{i}^x \epsilon_i + \]

- \( t_l^x \): center of the first-order error associated to the operation \( l \)
- \( t'_l^x \eta_l \): deviation on the first-order error associated to 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 \( t''_{i}^x \times y \epsilon_i \) comes from the multiplication of \( t_l^x \) by \( \alpha_i^y \epsilon_i \), and represents the uncertainty on the first-order error due to the uncertainty on the value, at label \( i \)
First-order errors

\[ e_1^x = \bigoplus_{l \in L} t_l^x + \bigoplus_{l \in L} t_l'^x \eta_l + \bigoplus_{i \in I} t_i''^x \varepsilon_i + \beta_0^x + \bigoplus_{p \in P} \beta_p^x \vartheta_p \]

- \( t_l^x \): center of the first-order error associated to the operation \( l \)
- \( t_l'^x \eta_l \): deviation on the first-order error associated to 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 \( \alpha_y \varepsilon_i \), and represents the uncertainty on the first-order error due to the uncertainty on the value, at label \( i \)
  - The multiplications of noise symbols \( \varepsilon_i \eta_l \) cannot be represented in our linear forms: we use a new affine form \( \vartheta_p \)
First example: an amazing scheme by Kahan and Muller

Compute, with $x_0 = 11/2.0$ and $x_1 = 61/11.0$, the sequence

$$x_{n+2} = 111 - \frac{1130 - \frac{3000}{x_n}}{x_{n+1}}$$

- If computed with real numbers, converges to 6. If computed with any approximation, converges to 100.
- Results with Fluctuat:
  - for $x_{10}$: finds the floating-point value equal to $f_{10} = 100$, with an error $e_{10}$ in $[-94.1261,-94.1258]$, and a real value $r_{10}$ in $[5.8812,5.8815]$
  - for $x_{100}$:
    - with default precision of the analysis (fp numbers with 60 bits mantissa), or even 400 mantissa bits numbers, finds $f_{100} = 100$, $e_{100} = \top$ and $r_{100} = \top$: indicates high unstability
    - with 500 mantissa bits numbers, finds $f_{100} = 100$, $e_{100} = -94$ and $r_{100} = 5.99...$
Second example: a non linear Newton scheme

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

```c
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++;
}
```

Sylvie Putot  MEASI, CEA-LIST/X/CNRS
We want to prove that for all A in [20,30], this Newton scheme terminates (in a reasonable number of iterations), for chosen stopping criterium $\varepsilon = e^{-10}$

- true in real numbers for all $\varepsilon$
- not obvious in finite precision (for example, there are some values of $A$ for which execution of this scheme but in simple precision float does not terminate)

Symbolic execution

- $A = 20.0 : i = 5, x_i = 5.0e-2 + [-2.82e-18,-2.76e-18]$
- $A = 30.0 : i = 9, x_i = 3.33e-2 + [-5.28e-18,6.21e-18]$

Static analysis for $A$ in $[20.0,30.0]$:

- Intervals: analysis does not prove termination
- Affine forms (with 10000 subdivisions): analysis finds $i \in [5,9]$, $x_i \in [3.33e-2,5.0e-2] + [-4.21e-13,4.21e-13]$
Third example: second-order filter

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

```c
double S,S0,S1,E,E0,E1;
S=0.0; S0=0.0;
E=__BUILTIN_DAED_DBETWEEN(0,1.0);
E0=__BUILTIN_DAED_DBETWEEN(0,1.0);
for (int 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 ;
}
Real enclosure tends to [-1.0907,2.7574]
```
Second-order filter

- Non relational analysis (interval arithmetic) : finds values and error equal to $\top$ (even increasing the number of bits of the mantissa)
- Relational analysis (affine arithmetic) on values and errors :
  
  Values in $[-1.09, 2.76]$  
  Error in $[-1.1e-14, 1.1e-14]$
References


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

- 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

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

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