

# Inferring numerical properties of embedded critical software

Sylvie Putot

with Eric Goubault, Franck Védrine and Karim Tekkal (Digiteo)

Laboratory for the Modelling and Analysis  
of Interacting Systems, CEA LIST

Session: Reliability and security of software

Digiteo Annual Forum - October 21, 2009  
Ecole Polytechnique

# Validation of numerical computations in critical software

- ▶ Embedded software do perform more and more numerical computations: filters, interpolators, approximation of elementary functions ...
- ▶ Conception designed in real numbers: what is the impact of a finite precision implementation ?
- ▶ What is a correct program using floating-point numbers ?
  - ▶ No run-time error, such as division by 0, overflow, etc
  - ▶ The program does compute something “not too far” from what is expected (=the result computed in real numbers)
  - ▶ No problematic control-flow difference between real and floating-point computation (same nb of iterations)
  - ▶ Can we also prove the algorithm correct (method error) ?



# Householder scheme for square root computation

# Floating-point numbers (IEEE 754 norm)

- ▶ Limited range and precision : potentially inaccurate results or run-time errors
- ▶ A few figures for simple precision normalized f.p. numbers :
  - ▶ largest  $\sim 3.40282347 * 10^{38}$
  - ▶ smallest positive  $\sim 1.17549435 * 10^{-38}$
  - ▶ max relative rounding error =  $2^{-23} \sim 1.19200928955 * 10^{-7}$
- ▶ Consequences:
  - ▶ potentially non intuitive representation error:  
 $\frac{1}{10} = 0.00011001100110011001100 \dots$  (binary)  
 $\Rightarrow \text{float}(\frac{1}{10}) = 0.1000000014901161194 \dots$  (decimal)
  - ▶ absorption :  $1 + 10^{-8} = 1$  in simple precision float
  - ▶ associative law not true :  $(-1 + 1) + 10^{-8} \neq -1 + (1 + 10^{-8})$
  - ▶ cancellation: loss of relative accuracy if subtracting close nbs



# In real world : costly or catastrophic examples

- ▶ 25/02/91: a Patriot missile misses a Scud and crashes on an american building : 28 dead.
  - ▶ 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
  - ▶ Drift on 100 hours  $\sim 0.34s$  : location error  $\sim 500m$
- ▶ Explosion of Ariane 5 in 1996 (conversion of a 64 bits float into a 16 bits integer : overflow)
- ▶ An index of the Vancouver stock exchange in 1982
  - ▶ truncated at each transaction : errors all have same sign
  - ▶ within a few months : lost half of its correct value
- ▶ Sinking of an offshore oil platform in 1992 : inaccurate finite element approximation

# Static Analysis

- ▶ Analysis of the source code, for a set of inputs and parameters, without executing it:
  - ▶ does the program always terminate?
  - ▶ does the program ever reach a bad state?
  - ▶ is there a possibility of run-time error, such as division by zero?
  - ▶ synchronization errors (deadlocks, data races)?
  - ▶ does the program compute accurately?
- ▶ Automatic static analysis vs other approaches:
  - ▶ Pen and paper proof tedious and error-prone
  - ▶ Testing / bug finding not exhaustive
  - ▶ Formal proof needs expertise, applicable for small programs
  - ▶ ... but static analysis may be over-approximated



# Problem of undecidability

- ▶ The ideal static analyzer (for run-time error) is
  - ▶ sound: if there is an error, the analyzer reports it
  - ▶ complete: if the analyzer reports an error, it is a genuine one
- ▶ But any interesting program property is undecidable in general:
  - ▶ in general choose sound (but not complete)
  - ▶ then focus on trade-off between performance and accuracy
  - ▶ static analysis by abstract interpretation (Cousot P. and R. 77)

# FLUCTUAT static analyzer: models float as real + error

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

$$f^x = 0.1 + 1.49e^{-9} [1]$$

$$f^y = 0.5$$

$$f^z = 0.6 + 1.49e^{-9} [1] + 2.23e^{-8} [3]$$

$$f^t = 0.06 + 1.04e^{-9} [1] + 2.23e^{-9} [3] - 8.94e^{-10} [4] - 3.55e^{-17} [ho]$$

⇒ Then abstraction for each term (real value and errors)



# Sound abstraction based on Affine Arithmetic

- ▶ The *real value* of variable  $x$  is represented by an affine form  $\hat{x}$  :

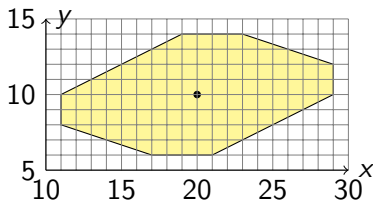
$$\hat{x} = x_0 + x_1\varepsilon_1 + \dots + x_n\varepsilon_n,$$

where  $x_i \in \mathbb{R}$  and the  $\varepsilon_i$  are independent symbolic variables with unknown value in  $[-1, 1]$ .

- ▶ Sharing  $\varepsilon_i$  between variables expresses *implicit dependency*: concretization as a zonotope

$$\hat{x} = 20 - 4\varepsilon_1 + 2\varepsilon_3 + 3\varepsilon_4$$

$$\hat{y} = 10 - 2\varepsilon_1 + \varepsilon_2 - \varepsilon_4$$



# Abstract domain based on affine arithmetic

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

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

- ▶ functional abstraction: link to the inputs via the noise symbols, allowing sensitivity analysis and worst case generation
- ▶ *Addition* is computed componentwise (no new noise symbol):

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

- ▶ *Non linear operations* : approximate linear form (Taylor expansion), new noise term for the approximation error
- ▶ Efficient join operator (on-going work for a better meet operator)

# Back to the Householder scheme

# Second order filters

# Some related work and tools for f.p. programs

Tools dedicated to the analysis of f.p. behavior:

- ▶ CADNA: roundoff error estimation by stochastic testing  
<http://www-anp.lip6.fr/cadna/>
- ▶ GAPPA: automatic proof generation of arithmetic properties  
<http://lipforge.ens-lyon.fr/www/gappa/>

Tools that take into account f.p. semantics:

- ▶ ASTREE: static analysis of run-time error  
<http://www.astree.ens.fr/>
- ▶ FRAMA-C: platform for source-code analysis of C software  
<http://frama-c.cea.fr/index.html>
- ▶ POLYSPACE: static analysis of run-time error  
[www.mathworks.com/products/polyspace/](http://www.mathworks.com/products/polyspace/)