# Automatic determination of numerical properties of software and systems

Eric Goubault and Sylvie Putot

Modelling and Analysis of Interacting Systems, CEA LIST

MASCOT-NUM 2012 Meeting, March 21-23, 2012

# Automatic validation of numerical programs

- Context: safety-critical programs
  - flight control or industrial install. (e.g. nuclear plants) control
  - long-term objective: scientific computing
  - guaranteed methods, that prove good behaviour or else try to give counter-examples
- Starting point: bound the error due to floating-point numbers
  - (deterministic) propagation of rounding errors in computations
  - can also propagate any uncertainty on inputs or parameters
- Now: towards functional proof of algorithms, hybrid systems
- A few words on an extension to stochastic uncertainties

# Static analysis of numerical programs

- Validate finite precision implementations: prove that the program computes something close to what is expected (in real numbers)
  - accuracy of results
  - behaviour of the program (control flow, number of iterations)
- But also validate algorithms: bound when possible the method/approximation error
  - ideally, find good match between method and implementation errors
- Automatically, given a source code, and for sets of (possibly uncertain) inputs and parameters: static analysis

# Householder scheme for square root computation

Householder

# Outline of the talk

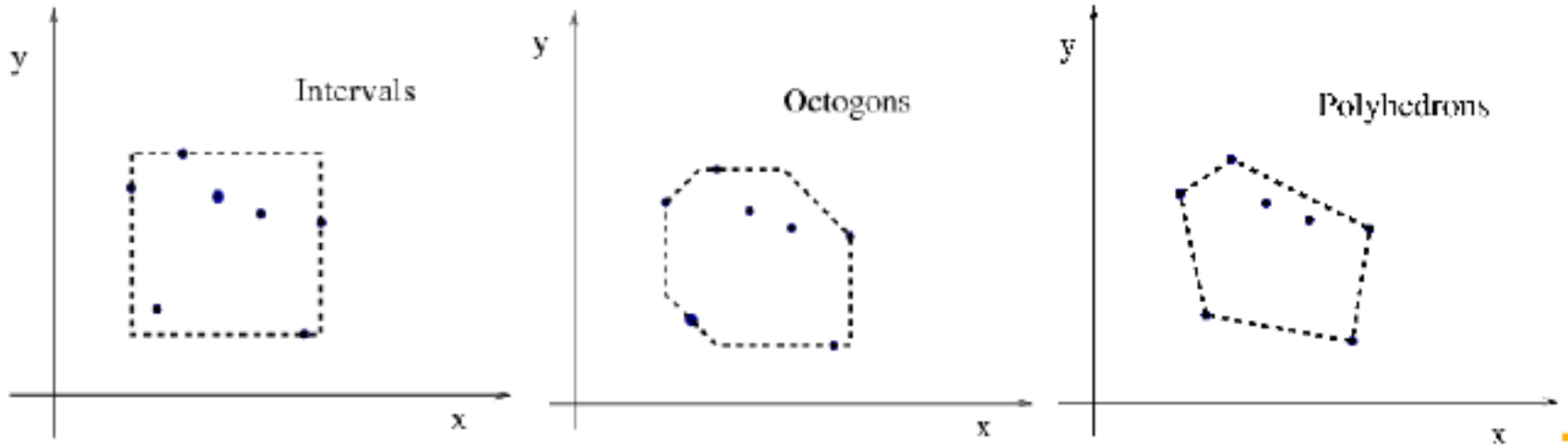
- Introduction to static analysis
- Static analysis of numerical programs
- Applications - the Fluctuat static analyzer
- First steps towards imprecise probabilities

# Static analysis

- Analysis of the source source, for a set of inputs and parameters, without executing it:
  - does the program always terminate; can it ever reach a bad state ?
  - is there a possibility of run-time error, such as division by zero?
  - worst case execution time?
  - synchronization errors (deadlocks, data races)?
  - does the program compute accurately?
- The ideal static analyzer (for run-time error for instance) 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, that expresses the property of interest

# Static analysis by abstract interpretation (Cousot 77)

- Choose properties of interest (for instance values of variables)
- Over-approximate them in an abstract lattice (partially ordered structure with least upper bounds/greatest lower bounds)




- Interpret computations in this lattice

# Example in intervals

- Program seen as a discrete dynamical system $X^{n+1} = F(Xn)$
  - based on a notion of control points in the program
  - equations describe how values of variables are collected at each control point, for all possible executions (collecting semantics)

```
void main() {
int x=[-100,50]; [1]
while [2] (x < 100) {
[3] x=x+1; [4]
} [5]
}
```

$$\begin{cases} x_0 = \top \\ x_1 = [-100, 50] \\ x_2 = x_1 \cup x_4 \\ x_3 = ]-\infty, 99] \cap x_2 \\ x_4 = x_3 + [1, 1] \\ x_5 = [100, +\infty[ \cap x_2 \end{cases}$$

- The sets of possible values of variables at control points are invariants of $F$, computed as the least fixpoint of the system
  - $F$ monotonic on a complete lattice, least fixpoint exists

# Invariants

- Invariants allow to conclude about the safety (for instance absence of run-time errors) of programs
- Computing the least fixpoint $X = F(X)$:
  - limit of the Kleene iteration (Jacobi/Gauss-Seidel like method) $X^0 = \bot$, $X^1 = F(X^0)$, ..., $X^{k+1} = X^k \cup F(X^k)$
    - with convergence acceleration to terminate in finite time
  - or policy iteration (Newton-like method)

# Outline of the talk

- Introduction to static analysis
- Static analysis of numerical programs
  - zonotopic abstract domains
  - modelling finite precision
- Applications - the Fluctuat static analyzer
- First steps towards imprecise probabilities

# Abstraction based on Affine Arithmetic (Stolfi 93)

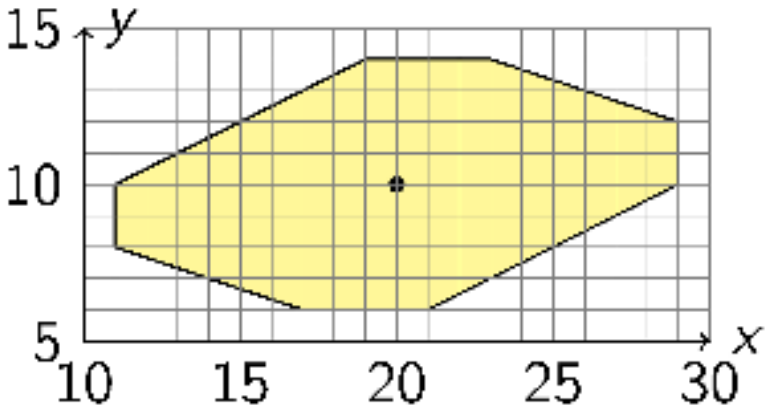
- The real value of variable $x$ is represented by an affine form $\hat{x}$ :

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

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

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

$$\begin{aligned} \hat{x} &= 20 - 4\varepsilon_1 + 2\varepsilon_3 + 3\varepsilon_4 \\ \hat{y} &= 10 - 2\varepsilon_1 + \varepsilon_2 - \varepsilon_4 \end{aligned}$$



# 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):

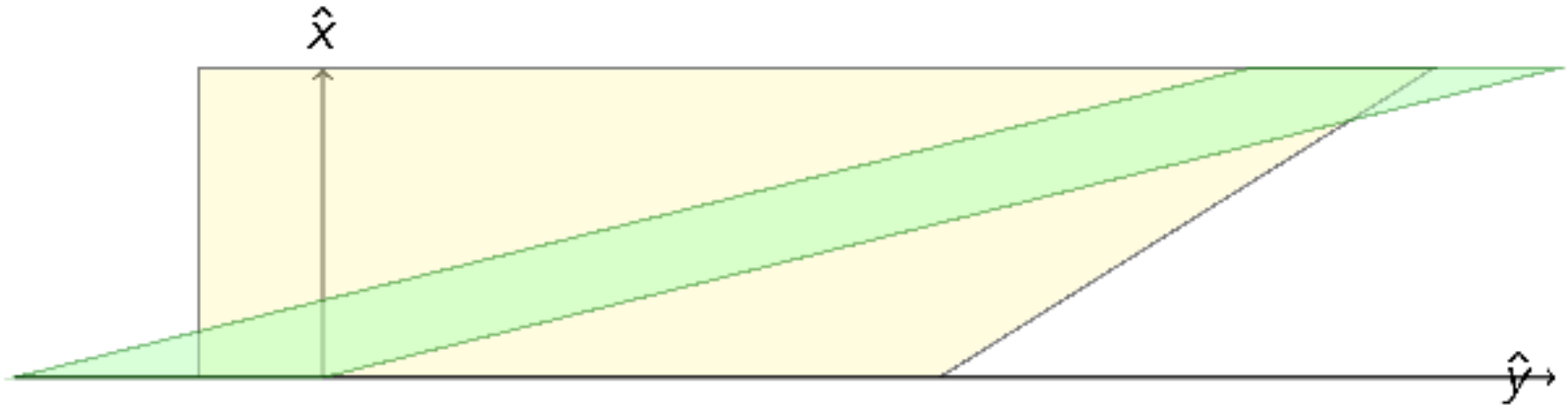
$$\hat{x} + \hat{y} = (\alpha_0^x + \alpha_0^y) + (\alpha_1^x + \alpha_1^y)\varepsilon_1 + \ldots + (\alpha_n^x + \alpha_n^y)\varepsilon_n$$

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

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

# Example

```
real x = [0,10];
real y = x*x - x;
```




Zonotope (green) is

$$\begin{aligned} x &= 5 + 5\varepsilon_1 \\ y &= (5 + 5\varepsilon_1)(5 + 5\varepsilon_1) - 5 - 5\varepsilon_1 \\ &= 20 + 45\varepsilon_1 + 25\varepsilon_1^2 = 20 + 45\varepsilon_1 + 25\,(0.5 + 0.5\varepsilon_2) \\ &= 32.5 + 45\varepsilon_1 + 12.5\varepsilon_2 \end{aligned}$$

Polyhedron (yellow) is

$-x + 10 \geq 0;\ y + 10 \geq 0;\ x \geq 0;\ 4x - y + 50 \geq 0$

# Affine arithmetic and static analysis

- Non linear operations: approximate linear form (Taylor expansion), new noise term for the approximation error
- close to Taylor models of low degree: because large ranges of values in general for static analysis
- possibly other polynomial approximations in the future

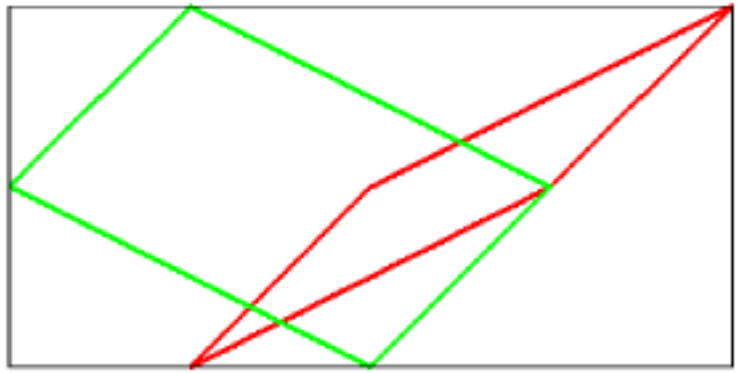
# Set-theoretical operations

- Partial order:
  - geometric inclusion on zonotopes augmented by slack variables for initial values of variables
  - functional order: preserves input-output relations
- Weaker structure than lattice: no least upper bound (join) or greatest lower bound (meet)
  - Join (to collect abstract values from different control flows): any upper bound of the union of zonotopes is sound
  - Meet (to interpret conditionals): we must over-approximate the intersection of zonotopes

# Example computation of a (m)ub

$$\begin{aligned} \hat{r}^x &= 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{r}^y &= 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{r}^u &= \varepsilon_1 + \varepsilon_2 \end{aligned}$$

$$\hat{r}^z = \hat{r}^x \cup \hat{r}^y = 2 + 0\varepsilon_1 + 1\varepsilon_2 + 3\varepsilon_U$$

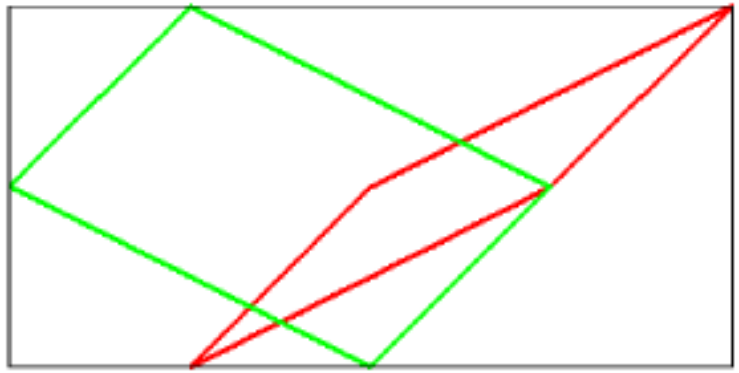


$\hat{r}^x$, $\hat{r}^y$ functions of $\hat{r}^u$

# Example computation of a (m)ub

$$\begin{aligned} \hat{r}^x &= 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{r}^y &= 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{r}^u &= \varepsilon_1 + \varepsilon_2 \end{aligned}$$

$$\hat{r}^z = \hat{r}^x \cup \hat{r}^y = 2 + 0\varepsilon_1 + 1\varepsilon_2 + 3\varepsilon_U$$



$\hat{r}^x$, $\hat{r}^y$ functions of $\hat{r}^u$

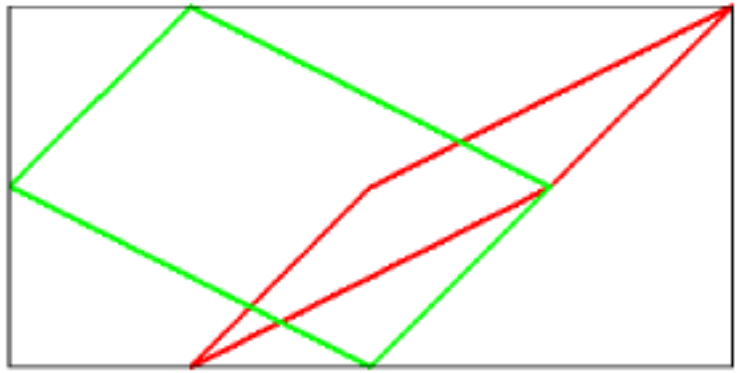
Keep "minimal common dependencies":

$$r_i^z = \underset{r_i^x \wedge r_i^y \leq \alpha \leq r_i^x \vee r_i^y}{\text{argmin}} |\alpha|, \ \forall i \geq 1$$

# Example computation of a (m)ub

$$\begin{aligned} \hat{r}^x &= 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{r}^y &= 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{r}^u &= \varepsilon_1 + \varepsilon_2 \end{aligned}$$

$$\hat{r}^z = \hat{r}^x \cup \hat{r}^y = 2 + 0\varepsilon_1 + 1\varepsilon_2 + 3\varepsilon_U$$



$\hat{r}^x$, $\hat{r}^y$ functions of $\hat{r}^u$

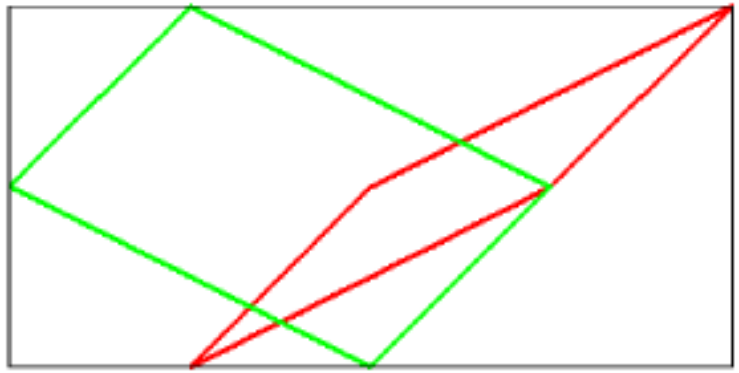
Keep "minimal common dependencies":

$$r_i^z = \underset{r_i^x \wedge r_i^y \leq \alpha \leq r_i^x \vee r_i^y}{\text{argmin}} |\alpha|, \ \forall i \geq 1$$

# Example computation of a (m)ub

$$\begin{aligned} \hat{r}^x &= 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{r}^y &= 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{r}^u &= \varepsilon_1 + \varepsilon_2 \end{aligned}$$

$$\hat{r}^z = \hat{r}^x \cup \hat{r}^y = 2 + 0\varepsilon_1 + 1\varepsilon_2 + 3\varepsilon_U$$



$\hat{r}^x$, $\hat{r}^y$ functions of $\hat{r}^u$

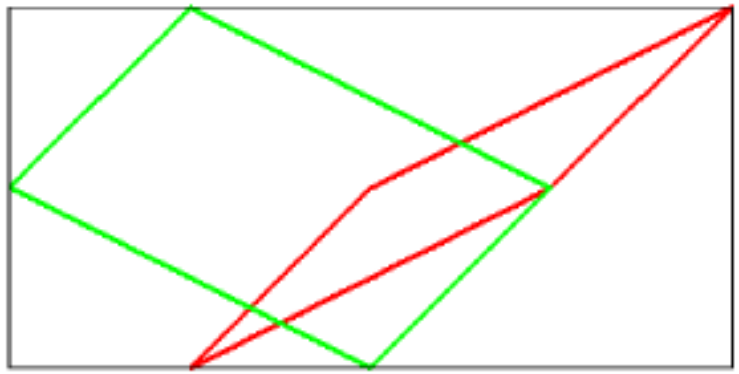
Keep "minimal common dependencies":

$$r_i^z = \underset{r_i^x \wedge r_i^y \leq \alpha \leq r_i^x \vee r_i^y}{\text{argmin}} |\alpha|, \ \forall i \geq 1$$

# Example computation of a (m)ub

$$\begin{aligned} \hat{r}^x &= 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{r}^y &= 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{r}^u &= \varepsilon_1 + \varepsilon_2 \end{aligned}$$

$\hat{r}^z = \hat{r}^x \cup \hat{r}^y = 2 + 0\varepsilon_1 + 1\varepsilon_2 + 3\varepsilon_U$



$\hat{r}^x$, $\hat{r}^y$ functions of $\hat{r}^u$

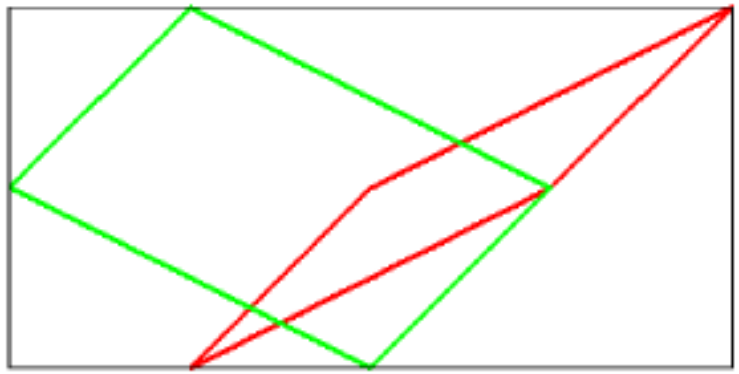
For each var, the concretization is the interval union of the concretizations ($\gamma(\hat{z}) = [-2, 6]$) :

$$\begin{aligned} &i) \quad r_0^z = mid(\gamma(\hat{r}^x) \cup \gamma(\hat{r}^y)) \\ &ii) \quad \beta^z = \sup \gamma(\hat{r}^x) \cup \gamma(\hat{r}^y) - r_0^z - \|\hat{r}^z\|_1 \end{aligned}$$

# Example computation of a (m)ub

$$\begin{aligned} \hat{r}^x &= 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{r}^y &= 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{r}^u &= \varepsilon_1 + \varepsilon_2 \end{aligned}$$

$\hat{r}^z = \hat{r}^x \cup \hat{r}^y = 2 + 0\varepsilon_1 + 1\varepsilon_2 + 3\varepsilon_U$



$\hat{r}^x$, $\hat{r}^y$ functions of $\hat{r}^u$

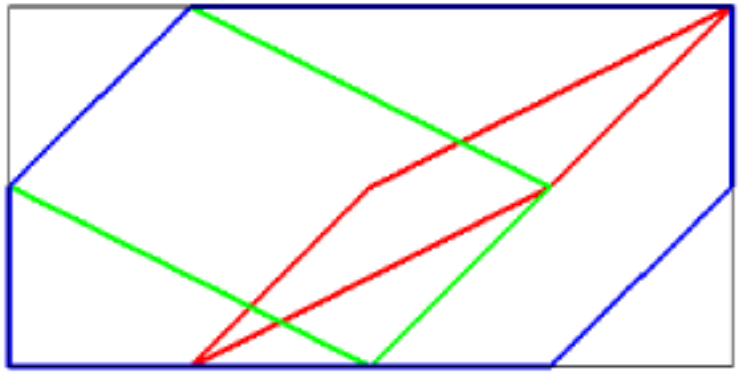
For each var, the concretization is the interval union of the concretizations ($\gamma(\hat{z}) = [-2, 6]$) :

$$\begin{aligned} &i) \quad r_0^z = mid(\gamma(\hat{r}^x) \cup \gamma(\hat{r}^y)) \\ &ii) \quad \beta^z = \sup \gamma(\hat{r}^x) \cup \gamma(\hat{r}^y) - r_0^z - \|\hat{r}^z\|_1 \end{aligned}$$

# Example computation of a (m)ub

$$\begin{aligned} \hat{r}^x &= 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{r}^y &= 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{r}^u &= \varepsilon_1 + \varepsilon_2 \end{aligned}$$

$\hat{r}^z = \hat{r}^x \cup \hat{r}^y = 2 + 0\varepsilon_1 + 1\varepsilon_2 + 3\varepsilon_U$



$\hat{r}^x$, $\hat{r}^y$ and $\hat{r}^z$ functions of $\hat{r}^u$

For each var, the concretization is the interval union of the concretizations ($\gamma(\hat{z}) = [-2, 6]$) :

$$\begin{aligned} &i) \quad r_0^z = mid(\gamma(\hat{r}^x) \cup \gamma(\hat{r}^y)) \\ &ii) \quad \beta^z = \sup \gamma(\hat{r}^x) \cup \gamma(\hat{r}^y) - r_0^z - \|\hat{r}^z\|_1 \end{aligned}$$

# Intersections: main idea, informally

```
real x = [0,10];
real y = 2*x;
if (y >= 10)
  y = x;
```

- Affine forms before tests: $x = 5 + 5\varepsilon_1$, $y = 10 + 10\varepsilon_1$
- Main idea to interpret tests:
  - translate the condition in the space of noise symbols: $\varepsilon_1 \geq 0$
  - abstract domain for the noise symbols: intervals, octagons, etc.
  - result may no longer be a zonotope
- Here $\varepsilon_1 \in [0, 1]$ in the `then` branch, implies $y \in [0, 10]$ at the end of the program

# 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:
    $\frac{1}{10} = 0.0001100110011001100110011 \cdots$ (binary)
    $\Rightarrow float(\frac{1}{10}) = 0.100000001490116119 4 \cdots$ (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 deads.
  - 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

# Our model for the analysis of numerical computations

- Aim: compute rounding errors and their propagation
  - for each variable, we compute $(f^x, r^x, e^x)$
  - then we will abstract each term (real value and errors)

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

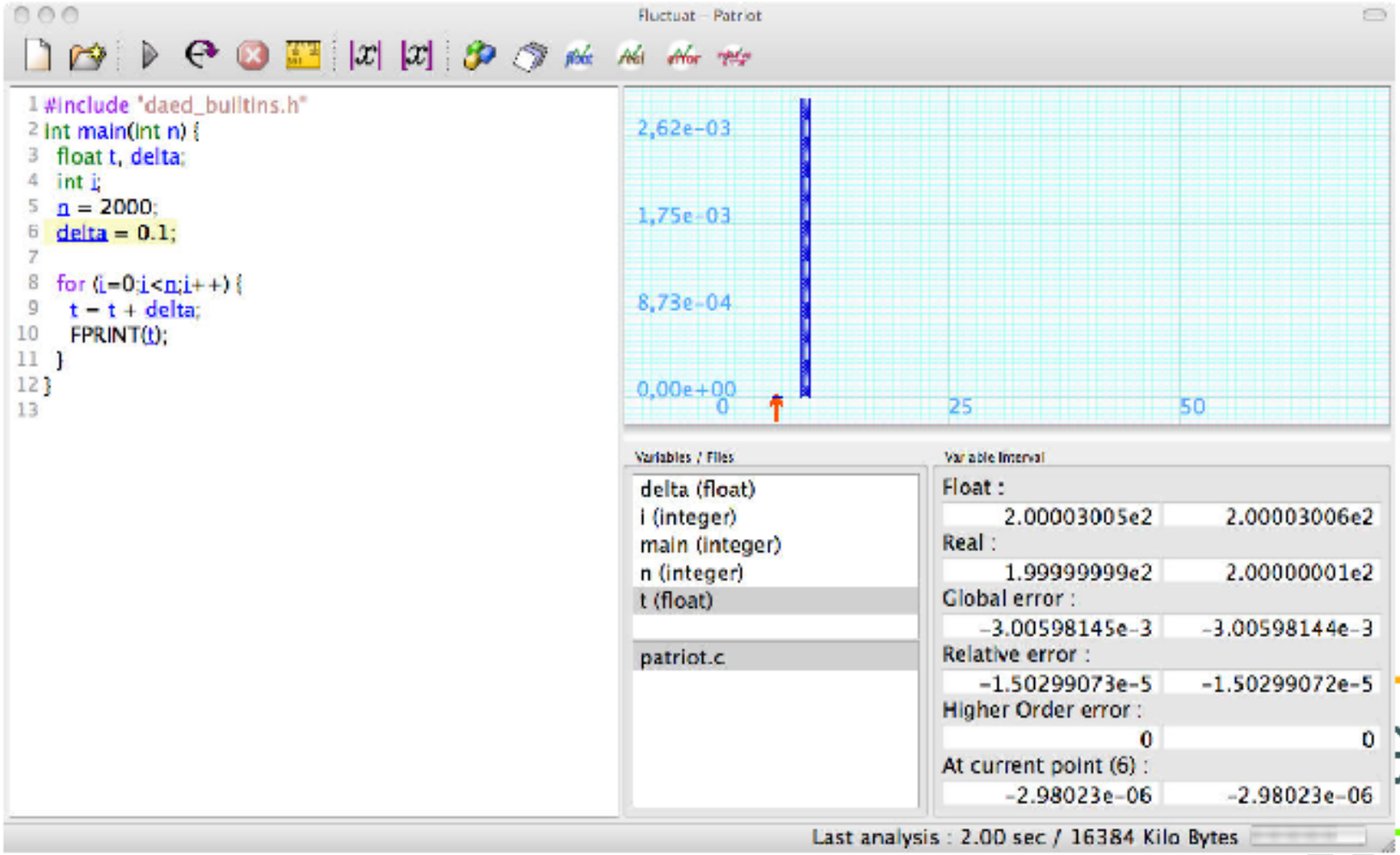
$$f^x = 0.1 + 1.49e^{-9}\ [1]$$

$$f^y = 0.5$$

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

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

# Example


Fluctuat - Patriot
1 #include "daed_builtins.h"
2 int main(int n) {
3 float t, delta;
4 int i;
5 n = 2000;
6 delta = 0.1;
7
8 for (i=0;i<n;i++) {
9 t = t + delta;
10 FPRINT(t);
11 }
12 }
13
2,62e-03
1,75e-03
8,73e-04
0,00e+00
0
25
50
Variables / Files
delta (float)
i (integer)
main (integer)
n (integer)
t (float)
patriot.c
Variable Interval
Float :
2.00003005e2
2.00003006e2
Real :
1.99999999e2
2.00000001e2
Global error :
-3.00598145e-3
-3.00598144e-3
Relative error :
-1.50299073e-5
-1.50299072e-5
Higher Order error :
0
0
At current point (6) :
-2.98023e-06
-2.98023e-06
Last analysis : 2.00 sec / 16384 Kilo Bytes

# Evolution of the error on t

# Towards an abstract model

- IEEE norm on f.p. numbers allows to prove properties:
  - elementary rounding error when rounding $r^x$ to $\uparrow_\circ r^x$:

$$\exists(\delta_r > 0, \delta_a > 0),\ |r^x - \uparrow_\circ r^x| \leq max(\delta_r |\uparrow_\circ r^x|, \delta_a)$$

  - the f.p. result of arithmetic elementary operations $+, -, \times, /, \sqrt{}$ is the rounded value of the real result

- For each variable $x$, a triplet $(f^x, r^x, e^x)$:
  - $r^x$ is an abstraction of the real (ideal) value of $x$
  - $f^x$ is an abstraction of the machine (finite precision) value of $x$
  - $e^x$ is an abstraction of the difference

- Abstraction of $r^x$, $e^x$ and $f^x$ using set-membership methods
  - the simplest: intervals
  - affine forms (zonotopes)

# Full abstraction

Abstract value: an interval and two affine forms $x = (\mathbf{f}^x, \hat{r}^x, \hat{e}^x)$:

$$\begin{aligned} \hat{r}^x &= r_0^x + \sum_i r_i^x \, \varepsilon_i^r \\ \hat{e}^x &= e_0^x + \sum_i e_i^x \, \varepsilon_i^r + \sum_l e_l^x \, \varepsilon_l^e \end{aligned}$$

- $\mathbf{f}^x = [\underline{f^x}, \overline{f^x}]$ bounds the finite prec value, $(\underline{f^x}, \overline{f^x}) \in \mathbb{F} \times \mathbb{F}$,
- $e_l^x \, \varepsilon_l^e$: uncertainty on the rounding error committed at point $l$ of the program (its center being in $e_0^x$), and its propagation through further computations,
- $e_i^x \, \varepsilon_i^r$: propagation of the uncertainty on value at point $i$, on the error term,
- dependency between errors and values partially modelled

# Outline of the talk

- Introduction to static analysis
- Static analysis of numerical programs
- Applications - the Fluctuat static analyzer
- First steps towards imprecise probabilities

# FLUCTUAT

- Takes source C code (most of ANSI C, except union types and malloc most notably), with assertions (for instance range of values and imprecision on input, but also range of gradient of evolution of values)
- Gives, fully automatically, characterization of ranges/errors, and describe the origins of errors: identification of pieces of code with numerical difficulties
- Is/has been used for a wide variety of codes (automotive, nuclear industry, aeronautics, aerospace) of size up to about 50000 LOCs (on laptop PCs 1Gb)
- Academic version available upon request

# Second order filters

Filters

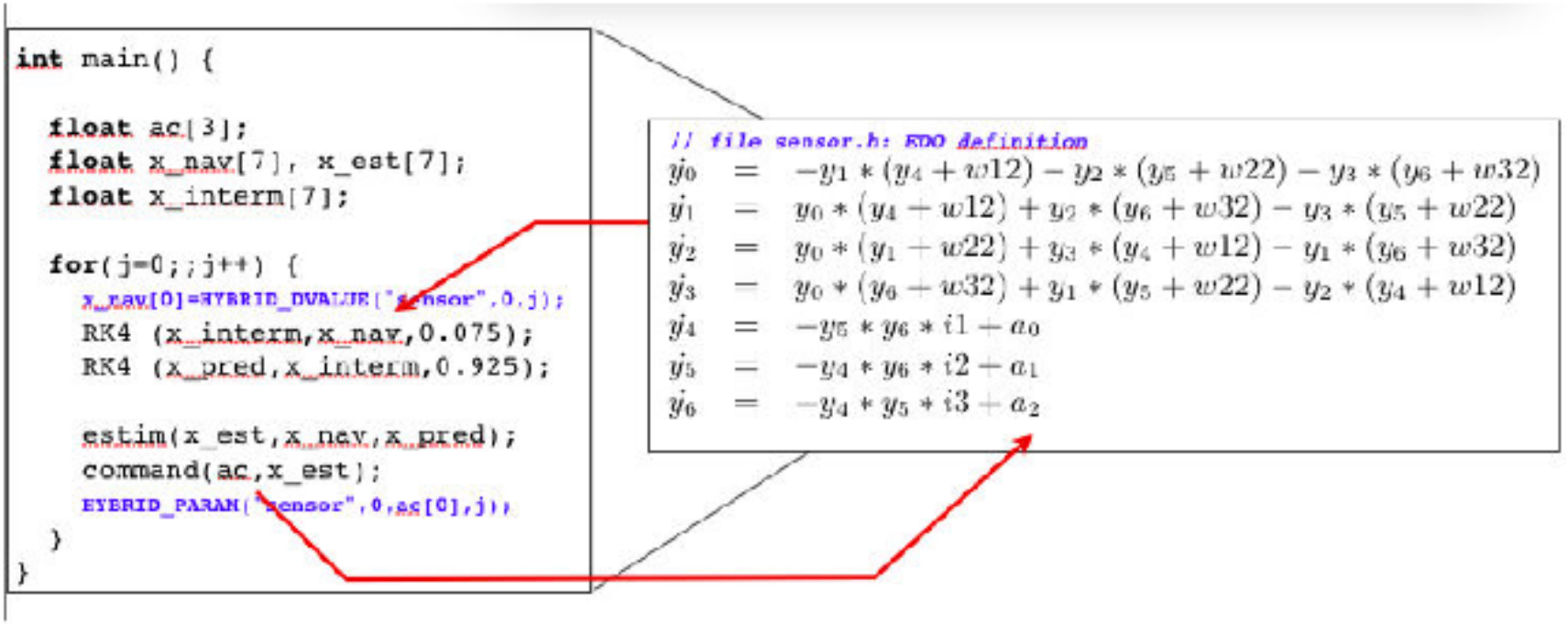
# Back to the Householder scheme

Householder

# First extension to hybrid systems analysis

- Classical program analysis: inputs given in ranges, possibly with bounds on the gradient between two values
  - Behaviour is often not realistic
- Hybrid systems analysis: analyze both physical environment and control software for better precision
  - Environment modelled by switched ODE systems
    - abstraction by guaranteed integration (the solver is guaranteed to over-approximate the real solution)
  - Interaction between program and environment modelled by assertions in the program
    - sensor reads a variable value at time $t$ from the environment,
    - actuator sends a variable value at time $t$ to the environment,
- Other possible use of guaranteed integration in program analysis: bound method error of ODE solvers

# Example: the ATV escape mechanism



```
int main() {

  float ac[3];
  float x_nav[7], x_est[7];
  float x_interm[7];

  for(j=0;;j++) {
    x_nav[0]=HYBRID_DVALUE("sensor",0.j);
    RK4 (x_interm,x_nav,0.075);
    RK4 (x_pred,x_interm,0.925);

    estim(x_est,x_nav,x_pred);
    command(ac,x_est);
    HYBRID_PARAM("sensor",0,ac[0],j);
  }
}
```

```
// file sensor.h: EDO definition
```

$$
\begin{aligned}
\dot{y}_0 &= -y_1 * (y_4 + w12) - y_2 * (y_5 + w22) - y_3 * (y_6 + w32) \\
\dot{y}_1 &= y_0 * (y_4 - w12) + y_2 * (y_6 + w32) - y_3 * (y_5 + w22) \\
\dot{y}_2 &= y_0 * (y_1 - w22) + y_3 * (y_4 + w12) - y_1 * (y_6 + w32) \\
\dot{y}_3 &= y_0 * (y_6 - w32) + y_1 * (y_5 + w22) - y_2 * (y_4 + w12) \\
\dot{y}_4 &= -y_5 * y_6 * i1 - a_0 \\
\dot{y}_5 &= -y_4 * y_6 * i2 - a_1 \\
\dot{y}_6 &= -y_4 * y_5 * i3 - a_2
\end{aligned}
$$

- Time is controlled by the program ($j$)
- Program changes parameters (`HYBRID_PARAM`: actuators) or mode (not here) of the ODE system
- Program reads from the environment(`HYBRID_DVALUE`: sensors) by calling the ODE guaranteed solver

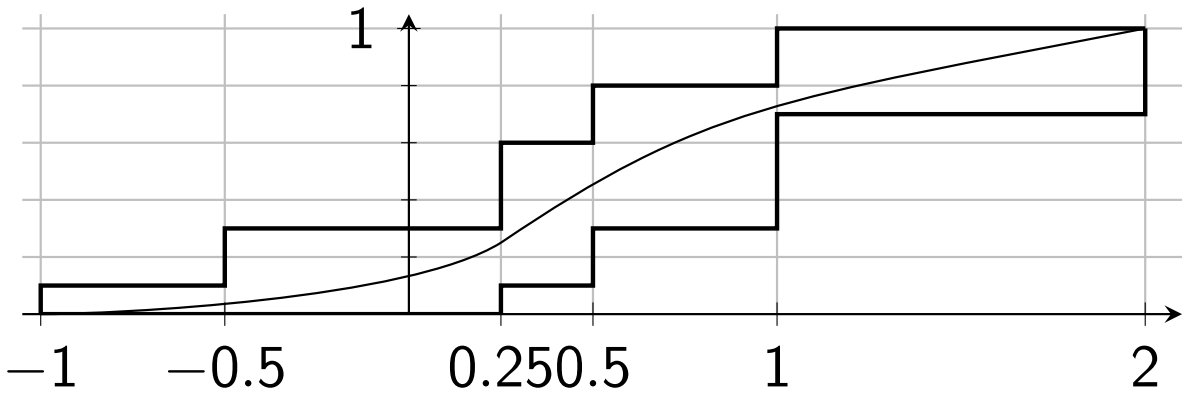
*Could demonstrate convergence towards the safe escape state.*

# Outline of the talk

- Introduction to static analysis
- Static analysis of numerical programs
- Applications - the Fluctuat static analyzer
- First steps towards imprecise probabilities

# Combining deterministic and probabilistic methods

- Inputs known in intervals or by probability distribution:
  - e.g. temperature distribution, but only a range for pressure
- More generally, inputs given by uncertain probabilities
  - Discrete p-boxes or Dempster-Shafer structures
  - Generalize probabilities and interval computations
  - Represent sets of probability distributions: between an upper and a lower Cumulative Distribution Function $P(X \leq x)$




- Less pessimistic but still guaranteed abstractions

# Arithmetic on Dempster-Shafer structures

- Starting point: arithmetic on discrete p-boxes or Dempster-Shafer structures not very efficient
  - rely on either independence or unknown dependency between variables
  - arithmetic for unknown dependency is costly and inaccurate

| | |
|---|---|
| X = input(); | New DS structure |
| Y = input(); | New (independent) DS structure |
| Z = X+Y; | Independent arithmetic |
| T = Z+X; | Must use unknown dependency(!) |

# Probabilistic affine forms

- Encode as much deterministic dependencies as possible by affine arithmetic
- Associate a DS structure to each noise symbol
- Only the non affine operations create DS structures (noise symbols) with dependency to some other symbols

| | |
|---|---|
| X = input(); | $X = \varepsilon_1$: new DS structure |
| Y = input(); | $Y = \varepsilon_2$: new (independent) DS structure |
| Z = X+Y; | $Z = \varepsilon_1 + \varepsilon_2$ |
| T = Z+X; | $T = 2\varepsilon_1 + \varepsilon_2$: DS of T uses independent arithmetic |

- Both more accurate and faster than direct DS arithmetic:
  - more accurate: relation exactly encoded
  - faster: operations on DS structures (the costly part) performed only when the DS of a variable is needed (end of the program or non linear operations)

# References: abstract domains, Fluctuat & cases studies

- Static Analysis of Numerical Algorithms, *SAS 2006 (Static Analysis Symposium)*
- The Zonotope Abstract Domain Taylor1+, *CAV 2009 (Computer Aided Verification)* and its extension in *CAV 2010*
- Static Analysis of Finite Precision Computations, *VMCAI 2011 (Verification, Model-Checking and Abstract Interpretation)*
- A generalization of P-boxes to affine arithmetic, *Computing, 2012*
- HybridFluctuat: A Static Analyzer of Numerical Programs within a Continuous Environment, *CAV 2009*
- Validation using Abstract Interpretation (with ESA, ASTRIUM SAS, ENS), *DASIA 2009 (DAta Systems In Aerospace Space Software)*
- Towards an industrial use of FLUCTUAT on safety-critical avionics software (with Airbus), *FMICS 2009*

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

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

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

Static analysis 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/

# Thanks for your attention!

Contact: {Eric.Goubault,Sylvie.Putot}@cea.fr