Static Analysis by Abstract Interpretation of Numerical Programs and Systems

Eric Goubault and Sylvie Putot

Cosynus team, LIX, Ecole Polytechnique

GT MEA, GdR MACS, 19th of March 2015

Example: Householder scheme for square root approx

Householder

What is "correctness" for numerical computations?

- No run-time error (division by 0, overflow, etc), see Astrée for instance
- The program computes a result close to what is expected
  - accuracy (and behaviour) of finite precision computations
  - method error

Context: safety-critical programs

- Typically flight control or industrial installation control (signal processing, instrumentation software)

Sound and automatic methods

- Guaranteed methods, that prove good behaviour or else try to give counter-examples
- Automatic methods, given a source code, and sets of (possibly uncertain) inputs and parameters

Abstract interpretation based static analysis

Computer-aided approaches to the problem of roundoff errors

Guaranteed computations or self-validating methods (dynamic): enclose the actual result as accurately as possible

- Set-based methods: interval (INTLAB library), affine arithmetic, Taylor model methods
- Specific solutions: verified ODE solvers, verified finite differences or finite element schemes

Error estimation: predict the behaviour of a finite precision implementation

- Dynamical control of approximations: stochastic arithmetic, CESTAC
- Uncertainty propagation by sensitivity analysis (Chaos polynomials)
- Formal proof, static analysis: (mostly) deterministic bounds on errors

Improve floating-point algorithms

- Specific (possibly proven correct) floating-point libraries (MPFR, SOLLYA)
- Automatic differentiation for error estimation and linear correction (CENA)
- Static-analysis based methods for accuracy improvement (SARDANA)

Automatic invariant synthesis

- Program seen as system of equations $X = F(X)$ on vectors of sets
  - 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)

Example

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

$$X = F(x)$$

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

Automatic invariant synthesis

- Program seen as a system of equations $X^{n+1} = F(X^n)$
- Want to compute reachable or invariant sets at control points
- Invariants allow to conclude about the safety (for instance absence of run-time errors) of programs
- Least fixpoint computation on partially ordered structure
  - classically computed as the limit of the Kleene (Jacobi) iteration

$$X^0 = \perp, X^1 = F(X^0), \ldots, X^{k+1} = X^k \cup F(X^k)$$

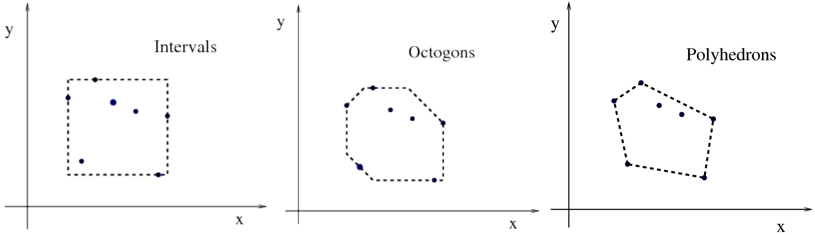
  - or policy iteration (Newton-like method - work with S. Gaubert et al. CAV 05, ESOP 10, LMCS 12 etc.)
- Generally not computable

Sound abstractions heavily relying on set-based methods

- Choose a computable abstraction that defines an over or under-approximation of set of values
- Need a partially ordered structure, with join and meet operators
- Transfer concrete fixpoint computation in the abstract world

Static analysis by abstract interpretation (Cousot 77)

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



- Interpret computations in this lattice

Back to our example

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

$$X = F(x)$$

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

First iterates (in fact, Gauss-Seidl)

$$\begin{pmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{pmatrix} \rightarrow \begin{pmatrix} -100, 50 \\ -100, 50 \\ -100, 50 \\ -99, 51 \\ \emptyset \end{pmatrix} \rightarrow \begin{pmatrix} -100, 50 \\ -100, 51 \\ -100, 51 \\ -99, 52 \\ \emptyset \end{pmatrix} \quad \ldots \quad \begin{pmatrix} -100, 50 \\ -100, 100 \\ -100, 99 \\ -99, 100 \\ 100, 100 \end{pmatrix}$$

Outline: around a family of zonotopic abstract domains

- parametrized zonotopes relying on Affine Arithmetic (Comba/Stolfi 92)
- an ordered structure for over-approximation of sets of real values
- a word on fixpoint computations
- finite precision analysis
- Fluctuat, examples
- variations and perspectives

Affine Arithmetic (Comba & Stolfi 93) for real-numbers abstraction

Affine forms

- Affine form for variable `x`:

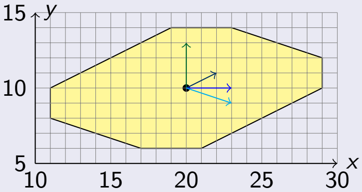
$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n, \ x_i \in \mathbb{R}$$

  where the ε_i are symbolic variables (*noise symbols*), with value in $[-1, 1]$.
- Sharing ε_i between variables expresses implicit dependency
- Interval concretization of affine form $\hat{x}$:

$$\left[x_0 - \sum_{i=0}^{n} |x_i|, x_0 + \sum_{i=0}^{n} |x_i|\right] = x_0 + [-\|(x_i)\|_1, \|(x_i)\|_1]$$

Geometric concretization as zonotopes (center symmetric polytopes)

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



Huge litterature - (dual) generator representation of a polytope!

- Assignment $x := [a, b]$ introduces a noise symbol:

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

- Addition/subtraction are exact:

$$\hat{x} + \hat{y} = (x_0 + y_0) + (x_1 + y_1)\varepsilon_1 + \ldots + (x_n + y_n)\varepsilon_n$$

- Non linear operations : approximate linear form, new noise term bounding the approximation error

$$\hat{x} \times \hat{y} = x_0 y_0 + \sum_{i=0}^{n}(x_0 y_i + x_i y_0)\varepsilon_i + \left(\sum_{1 \le i \ne j \le n} | x_i y_j |\right)\varepsilon_{n+1}$$

(better formulas including SDP computations of the new term)

- Close to Taylor models of low degree : low time complexity! and easy to implement on a finite-precision machine (for general polyhedra, see Miné APLAS 2008)

Very appealing model...part of a bigger picture

Taylor models approximate variables values by polynomial plus remainder:

$$f(x_1, \ldots, x_n) = f(0) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(0) x_i + \ldots$$

(Chapoutot Ph.D. thesis, Zumkeller version with COQ, Berz' COSY for ODE guaranteed integration, Sriram's FLOW* etc.)

Very appealing model...part of a bigger picture

Taylor models approximate variables values by polynomial plus remainder:

$$f(x_1,\ldots,x_n) = f(0) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(0)x_i + \sum_{i,j=1}^{n} \frac{1}{2}\frac{\partial^2 f}{\partial x_i \partial x_j}(0)x_i x_j + \ldots$$

(Chapoutot Ph.D. thesis, Zumkeller version with COQ, Berz' COSY for ODE guaranteed integration, Sriram's FLOW* etc.)

Taylor models

Very appealing model...part of a bigger picture

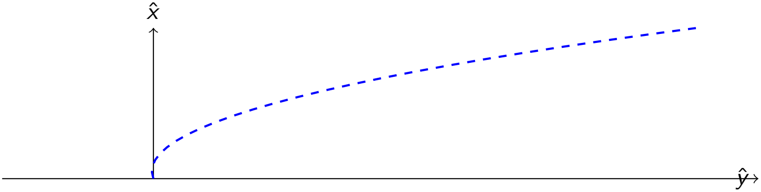
Taylor models approximate variables values by polynomial plus remainder:

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i \varepsilon_i + \sum_{i,j=1}^{n} x_{i,j} \varepsilon_i \varepsilon_j + [R]$$

(Chapoutot Ph.D. thesis, Zumkeller version with COQ, Berz' COSY for ODE guaranteed integration, Sriram's FLOW* etc.)

A simple example: functional interpretation

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

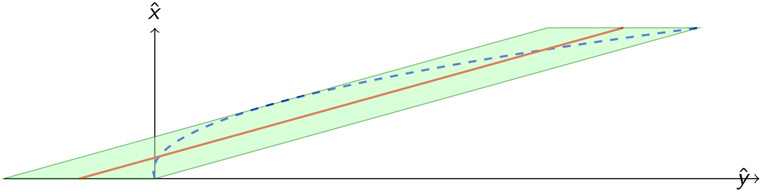


Abstraction of x: $x = 5 + 5\varepsilon_1$
Abstraction of function $x \to y = x^2 - x$ as

$$y \quad = \quad 32.5 + 50\varepsilon_1 + 12.5\eta_1$$

A simple example: functional interpretation

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



Abstraction of x: $x = 5 + 5\varepsilon_1$
Abstraction of function $x \to y = x^2 - x$ as

$$\begin{aligned} y &= 32.5 + 50\varepsilon_1 + 12.5\eta_1 \\ &= -17.5 + 10x + 12.5\eta_1 \end{aligned}$$

Set operations on affine sets / zonotopes: meet

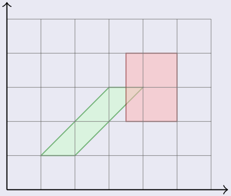
Reminder

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

$$X = F(x)$$

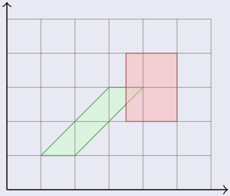
$$\begin{cases} x_1 &= [-100, 50] \\ x_2 &= x_1 \cup x_4 \\ x_3 &=]-\infty, 99] \cap x_2 \\ x_4 &= x_3 + 1 \\ x_5 &= [100, +\infty[\cap x_2 \end{cases}$$

Intersection of zonotopes are not zonotopes!



Set operations on affine sets / zonotopes: meet

Intersection of zonotopes are not zonotopes!

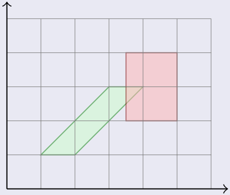


Interpreting tests (CAV 2010)

- Translate the condition on noise symbols: constrained affine sets
- Abstract domain for the noise symbols: intervals, octagons, etc.
- Equality tests are interpreted by the substitution of one noise symbol of the constraint (cf summary instantiation for modular analysis)

Set operations on affine sets / zonotopes: meet

Intersection of zonotopes are not zonotopes!



Example

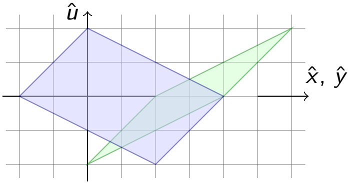
`real x = [0,10]; real y = 2*x; if (y >= 10) y = x;`

- Affine forms before tests: $x = 5 + 5\varepsilon_1$, $y = 10 + 10\varepsilon_1$
- In the `if` branch $\varepsilon_1 \geq 0$: condition acts on both x and y

Arithmetic operations carry over nicely to this logical/reduced product

Join operator

$$\begin{pmatrix} \hat{x} = 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{u} = 0 + \varepsilon_1 + \varepsilon_2 \end{pmatrix} \cup \begin{pmatrix} \hat{y} = 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{u} = 0 + \varepsilon_1 + \varepsilon_2 \end{pmatrix} = \begin{pmatrix} \hat{x} \cup \hat{y} & = & 2 + \varepsilon_2 + 3\eta_1 \\ \hat{u} & = & 0 + \varepsilon_1 + \varepsilon_2 \end{pmatrix}$$



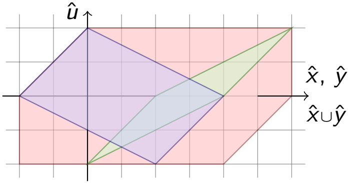
Construction (low complexity!: $\mathcal{O}(n \times p)$)

- Keep "minimal common dependencies"

$$z_i = \underset{x_i \wedge y_i \leq r \leq x_i \vee y_i}{\text{argmin}} |r|, \ \forall i \geq 1$$

Join operator

$$\begin{pmatrix} \hat{x} = 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{u} = 0 + \varepsilon_1 + \varepsilon_2 \end{pmatrix} \cup \begin{pmatrix} \hat{y} = 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{u} = 0 + \varepsilon_1 + \varepsilon_2 \end{pmatrix} = \begin{pmatrix} \hat{x} \cup \hat{y} & = & 2 + \varepsilon_2 + 3\eta_1 \\ \hat{u} & = & 0 + \varepsilon_1 + \varepsilon_2 \end{pmatrix}$$



Construction (low complexity!: $\mathcal{O}(n \times p)$)

- Keep "minimal common dependencies"

$$z_i = \operatorname*{argmin}_{x_i \wedge y_i \leq r \leq x_i \vee y_i} |r|, \ \forall i \geq 1$$

- For each dimension, concretization is the interval union of the concretizations: $\gamma(\hat{x} \cup \hat{y}) = \gamma(\hat{x}) \cup \gamma(\hat{y})$
- A more precise upper bound: NSAD 2012

General result on recursive linear filters, pervasive in embedded programs:

$$x_{k+n+1} = \sum_{i=1}^{n} a_i x_{k+i} + \sum_{j=1}^{n+1} b_j e_{k+j}, \quad e_l \in [m, M]$$

- Concrete scheme has bounded outputs iff zeros of $x^n - \sum_{i=0}^{n-1} a_{i+1} x^i$ have modulus stricty lower than 1.
- Then our Kleene iteration (with some uncyclic unfolding q) converges towards a finite over-approximation of the outputs

$$\hat{X}_i = \hat{X}_{i-1} \cup F^q(E_i, \ldots, E_{i-k}, \hat{X}_{i-1}, \ldots, \hat{X}_{i-k})$$

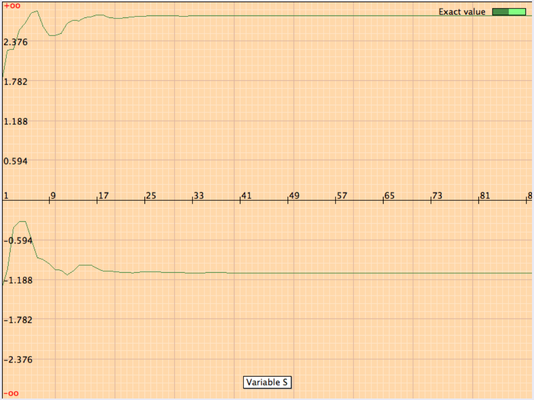
  in finite time
  - The abstract scheme is a perturbation (by the join operation) of the concrete scheme
  - Proof uses: for each dimension $\gamma(\hat{x} \cup \hat{y}) = \gamma(\hat{x}) \cup \gamma(\hat{y})$ and F^q is contracting "enough" for some q

Generalization to some recurrent polynomial schemes

A simple order 2 filter

$$S_{n+2} = 0.7E_{n+2} - 1.3E_{n+1} + 1.1E_n + 1.4S_{n+1} - 0.7S_n$$

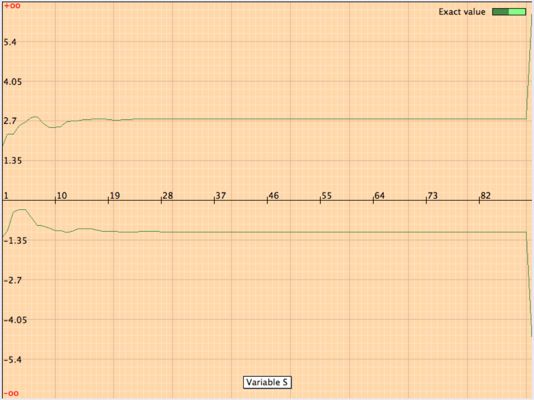
Step 0: initial unfolding (10)+first cyclic unfolding (80) - first join



A simple order 2 filter

$$S_{n+2} = 0.7E_{n+2} - 1.3E_{n+1} + 1.1E_n + 1.4S_{n+1} - 0.7S_n$$

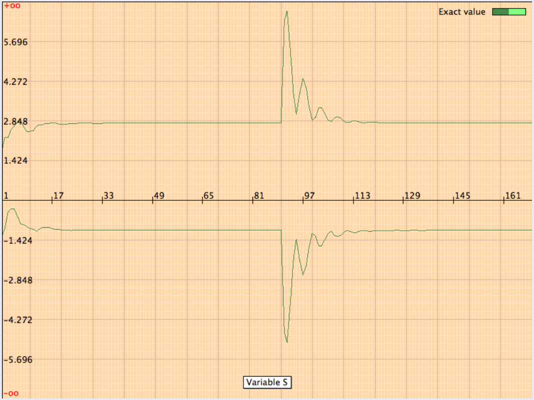
Step 1: After first join, perturbation of the original numerical scheme!



A simple order 2 filter

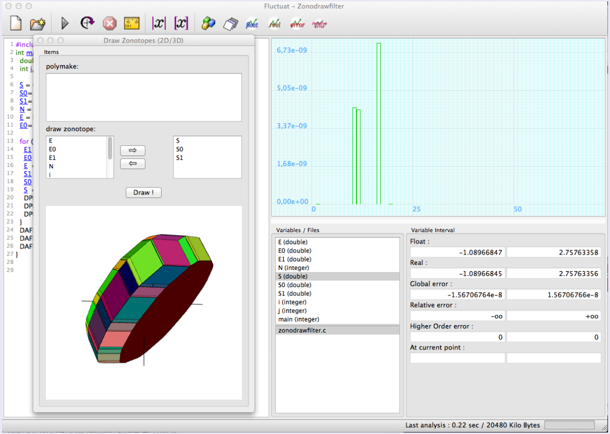
$$S_{n+2} = 0.7E_{n+2} - 1.3E_{n+1} + 1.1E_n + 1.4S_{n+1} - 0.7S_n$$

Step 2: second cyclic unfolding, contracting back - second join and post-fixpoint



A simple order 2 filter

$$S_{n+2} = 0.7E_{n+2} - 1.3E_{n+1} + 1.1E_n + 1.4S_{n+1} - 0.7S_n$$



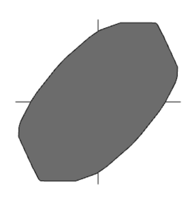
A simple order 2 filter

$$S_{n+2} = 0.7E_{n+2} - 1.3E_{n+1} + 1.1E_n + 1.4S_{n+1} - 0.7S_n$$

- This is a polyhedral approximation of the classical ellipsoidal invariant
- May be inefficient, for convergence, q of the order of

$$-\frac{\log 2}{\log \sup_{\lambda eigenvalue} \mid \lambda \mid}$$

  (here spectral radius of 0.84, but q ~15 for 0.95, 138 for 0.995 etc.)
- Although several ten thousands of symbols is manageable, there is a way to hack the domain to describe mixed zonotopic/ellipsoidal invariants, see NSV 2011



Ellipsoidal domains

Long history

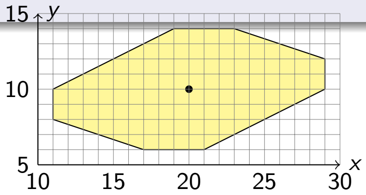
Kurzhanski in Control Theory (1991), Feret (ESOP 2004), Cousot (VMCAI 2005), Adjé et al. (ESOP 2010), Gawlitza et al. (SAS 2010), Garoche et al. (HSCC 2012) etc.

Simple add-on

Extend affine forms to ellipsoidal forms ? Replace...

$$\hat{x} = x_0 + \sum_{i=1}^{n} \mathbf{x}_i \varepsilon_i, \text{ with } \|\varepsilon\|_\infty = \sup_{i=1,\ldots,n} |\varepsilon_i| \leq 1$$

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



Ellipsoidal domains

Long history

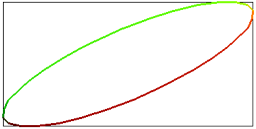
Kurzhanski in Control Theory (1991), Feret (ESOP 2004), Cousot (VMCAI 2005), Adjé et al. (ESOP 2010), Gawlitza et al. (SAS 2010), Garoche et al. (HSCC 2012) etc.

Simple add-on

Extend affine forms to ellipsoidal forms ? Replace...

$$\hat{x} = x_0 + \sum_{i=1}^{n} \mathbf{x}_i \varepsilon_i, \text{ with } \|\varepsilon\|_2 = \sqrt{\sum_{i=1}^{n} \varepsilon_i^2} \leq 1$$

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



Ellipsoidal domains

Long history

Kurzhanski in Control Theory (1991), Feret (ESOP 2004), Cousot (VMCAI 2005), Adjé et al. (ESOP 2010), Gawlitza et al. (SAS 2010), Garoche et al. (HSCC 2012) etc.

Simple add-on

Extend affine forms to ellipsoidal forms ? Replace...

$$\hat{x} = x_0 + \sum_{i=1}^{n} \mathbf{x}_i \varepsilon_i, \text{ with } \|\varepsilon\|_2 = \sqrt{\sum_{i=1}^{n} \varepsilon_i^2} \leq 1$$

Functional order

$X \subseteq Y$ if and only if for all $t \in \mathbb{R}^p$

$$\|(C^X - C^Y)t\|_2 \leq \|P^Y t\|_2 - \|P^X t\|_2$$

This is the right order for functional abstractions (Lorenz cone!)
On going work with Maxplus (S. Gaubert, X. Allamigeon) and Nikolas Stott (Ph.D.)

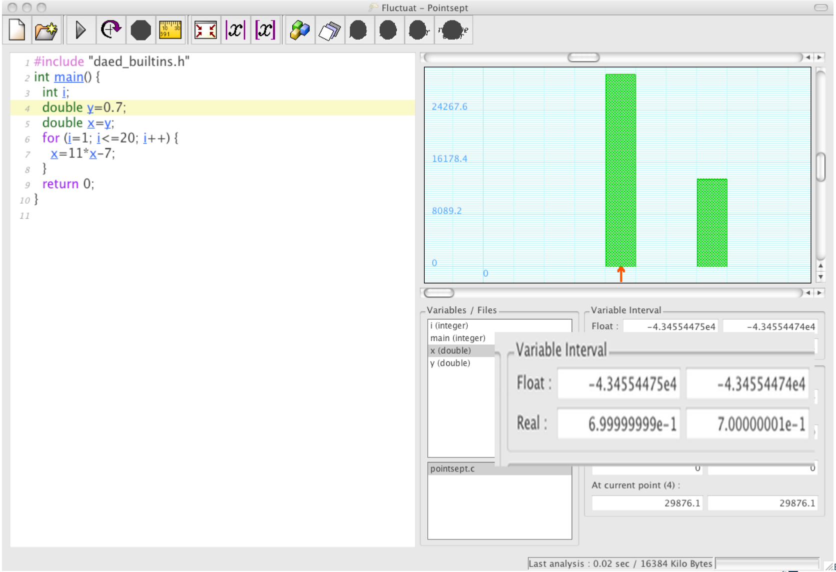
FLUCTUAT: concrete semantics

- IEEE 754 norm on f.p. numbers specifies the rounding error (same is feasible for fixed point semantics)
- Aim: compute rounding errors and their propagation
  - we need the floating-point values
  - relational (thus accurate) analysis more natural on real values
  - 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]
```

$$
\begin{aligned}
f^x &= 0.1 + 1.49e^{-9}\ [1] \\
f^y &= 0.5 \\
f^z &= 0.6 + 1.49e^{-9}\ [1] + 2.23e^{-8}\ [3] \\
f^t &= 0.06 + 1.04e^{-9}\ [1] + 2.23e^{-9}\ [3] - 8.94e^{-10}\ [4] - 3.55e^{-17}\ [ho]
\end{aligned}
$$

Example (Fluctuat)



Abstract value

- For each variable x, a triplet (f^x, r^x, e^x):
  - Interval $\mathbf{f}^x = [\underline{f^x}, \overline{f^x}]$ bounds the finite prec value, $(\underline{f^x}, \overline{f^x}) \in \mathbb{F} \times \mathbb{F}$,
  - Affine forms for real value and error; for simplicity no η symbols

$$f^x = \underbrace{(\alpha_0^x + \bigoplus_i \alpha_i^x \varepsilon_i^r)}_{\text{real value}} + (\underbrace{e_0^x}_{\text{center of the error}} + \underbrace{\bigoplus_l e_l^x \varepsilon_l^e}_{\text{uncertainty on error due to point } l} + \underbrace{\bigoplus_i m_i^x \varepsilon_i^r}_{\text{propag of uncertainty on value at pt } i})$$

- Constraints on noise symbols (interval + equality constraints)
  - for finite precision control flow
  - for real control flow

Back to the Householder scheme

Householder

Second order filters

Filters

Extension: 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

```c
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

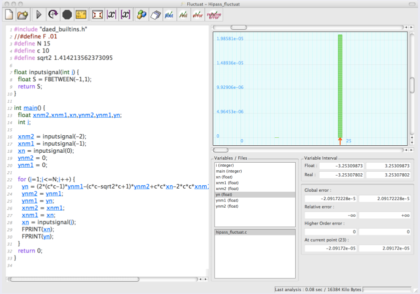
Could demonstrate convergence towards the safe escape state (CAV 2009, DASIA 2009 with Olivier Bouissou).

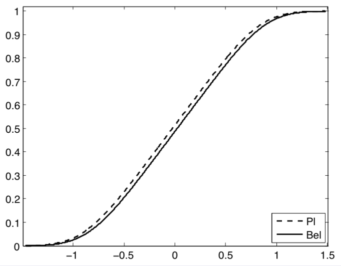
ATV

Keep same parameterization $x = \sum_i x_i \varepsilon_i$ but with

- Interval coefficients x_i: generalized affine sets for under-approximation
    - under-approximation: sets of values of the outputs, that are sure to be reached for some inputs in the specified ranges
    - interval coefficients x_i, noise symbols in generalized intervals ($\boldsymbol{\varepsilon}_{\mathbf{i}} = [-1, 1]$ or $\boldsymbol{\varepsilon}^*_{\mathbf{i}} = [1, -1]$), Kaucher arithmetic extends classical interval arithmetic (SAS 2007, HSCC 2014 with M. Kieffer)
- Noise symbols ε_i coding sets of probability distributions:
    - probabilistic affine forms: ε_i take values in probability boxes (Computing 2012, with O. Bouissou, J. Goubault-Larrecq)

Prove that dangerous worst case occur with very low probability





- Deterministic analysis (left): outputs in [-3.25,3.25] (exact)
- Mixed probabilistic/deterministic analysis (right): outputs in [-3.25,3.25], and in [-1,1] with very strong probability (in fact, very close to a Gaussian distribution)

Quite some success up to now (now agreement between CEA and X)

- On industrial code (up to 100KLoc), mostly on control code (nuclear plants, automotive industry, aeronautics and space industry etc.)
- Used by Airbus for the A350
- see e.g. FMICS 2007, 2009, DASIA 2009

Still...

- Rather simple numerical computations: linear recursive filters, linear control, mathematical librairies (at the exception of Astrium's ATV)
- What about cyber-physical systems, i.e. distributed control programs?
- What about simulation programs such as finite element methods etc.?
- A good start: Lanczos/conjugate gradient methods for solving linear systems, at the heart of such implementation

This is still a long term goal...

Goal and difficulties

- link with mathematical studies of schemes in finite precision (Wilkinson 1965, Paige 1971, Meurant 2006, Demmel ICM talk 2002 etc), viewed as perturbed schemes
- specific problems to solve (large arrays with specific [sparse] patterns)

Practically speaking?

- Difficult to design specific abstract domains for each one of the numerical codes
- Interaction with provers is under study (e.g. FRAMA-C)
- Idea is: proof in floating-point numbers is a perturbation of the proof in reals, in general (partially) formally available at design phase

In the long run...

Many more “details” to solve...

- Numerical simulation codes are parallel, implement fault-tolerant mechanisms, run petaflopic operations, some algorithms are randomized (e.g. Monte-Carlo codes etc.)
- Running on complex multicore and GPU architectures:
  - Evaluation order highly dependent of schedules, weak memory models etc. : recall, numerical properties depend on the evaluation order!

In the long run...

Many more "details" to solve...

- Numerical simulation codes are parallel, implement fault-tolerant mechanisms, run petaflopic operations, some algorithms are randomized (e.g. Monte-Carlo codes etc.)
- Running on complex multicore and GPU architectures:
    - Evaluation order highly dependent of schedules, weak memory models etc. : recall, numerical properties depend on the evaluation order!
- Real embedded systems are redundant, distributed, less and less synchronous, hybrid, manage probabilistic events and data etc.

Any questions?

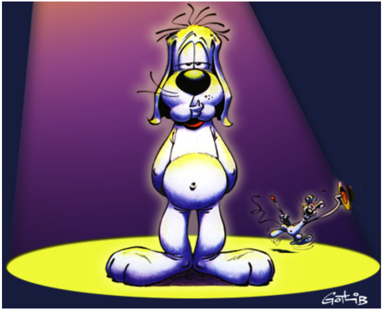
Tried to show that "explicit" (generator-based) (sub-)polyhedric domains such as zonotopes...

- have low complexity
- can be studied as numerical schemes of their own
- can easily be extended in order to deal with other or more refined properties: finite precision semantics, polynomial abstractions, under-approximations, hybrid systems analysis, probabilistic systems etc.

One goal is to carry on all the way to very complex parallel numerical codes and cyber-physical systems on modern architectures...!

Any questions?

!



Mean-value theorem (à la Goldsztejn 2005)

Let $f : \mathbb{R}^n \to \mathbb{R}$ differentiable, $(t_1, \ldots, t_n)$ a point in $[-1, 1]^n$ and $\mathbf{\Delta_i}$ such that

$$\left\{ \frac{\partial f}{\partial \varepsilon_i}(\varepsilon_1, \ldots, \varepsilon_i, t_{i+1}, \ldots, t_n),\ \varepsilon_i \in [-1, 1] \right\} \subseteq \mathbf{\Delta_i}.$$

Then

$$\tilde{f}(\varepsilon_1, \ldots, \varepsilon_n) = f(t_1, \ldots, t_n) + \sum_{i=1}^{n} \mathbf{\Delta_i}(\varepsilon_i - t_i),$$

is interpretable in the following way :

- if $\tilde{f}(\boldsymbol{\varepsilon}_\mathbf{1}^*, \ldots, \boldsymbol{\varepsilon}_\mathbf{n}^*)$, computed with Kaucher arithmetic, is an improper interval, then pro $\tilde{f}(\boldsymbol{\varepsilon}_\mathbf{1}^*, \ldots, \boldsymbol{\varepsilon}_\mathbf{n}^*)$ is an under-approx of $f(\boldsymbol{\varepsilon}_\mathbf{1}, \ldots, \boldsymbol{\varepsilon}_\mathbf{n})$.
- $\tilde{f}(\boldsymbol{\varepsilon}_\mathbf{1}, \ldots, \boldsymbol{\varepsilon}_\mathbf{n})$ is an over-approx of $f(\boldsymbol{\varepsilon}_\mathbf{1}, \ldots, \boldsymbol{\varepsilon}_\mathbf{n})$.

Generalized affine forms

- Affine forms with interval coefficients, defined on the ε_i (no η_j symbols)
- Under-approximation by over-approximation of dependencies
- Joint use of under-/over-approximation: quality of analysis results
- Extract scenarios giving extreme values

Example

$$f(x) = x^2 - x \;\text{ when } x \in [2,3] \;\text{ (real result } [2,6])$$

- Affine form

$$x = 2.5 + 0.5\varepsilon_1, \quad f^{\varepsilon}(\varepsilon_1) = (2.5 + 0.5\varepsilon_1)^2 - (2.5 + 0.5\varepsilon_1)$$

- Bounds on partial derivative

$$\frac{\partial f^{\varepsilon}}{\partial \varepsilon_1}(\varepsilon_1) = 2 * 0.5 * (2.5 + 0.5\varepsilon_1) - 0.5 \subseteq [1.5, 2.5]$$

- Mean value theorem with $t_1 = 0$

$$\tilde{f}^{\varepsilon}(\varepsilon_1) = 3.75 + [1.5, 2.5]\varepsilon_1$$

Under-approximating concretization
$3.75 + [1.5, 2.5][1, -1] = 3.75 + [1.5, -1.5] = [5.25, 2.25]$
Over-approximating concretization
$3.75 + [1.5, 2.5][-1, 1] = 3.75 + [-2.5, 2.5] = [1.25, 6.25]$

- Affine arithmetic (over-approximation)

$$x^2 - x = [3.75, 4] + 2\varepsilon_1 \;\text{ (concretization } [1.75, 6])$$

Square-root algorithm (Householder method)

```
double Input, x, xp1, residue, shouldbezero;
double EPS = 0.00002;

Input = __BUILTIN_DAED_DBETWEEN(16.0,20.0);
x = 1.0/Input; xp1 = x; residue = 2.0*EPS;
while (fabs(residue) > EPS) {
  xp1 = x*(1.875+Input*x*x*(-1.25+0.375*Input*x*x));
  residue = 2.0*(xp1-x)/(x+xp1);
  x = xp1;
}
shouldbezero = x*x-1.0/Input;
```

- With 32 subdivisions of the input
  - Stopping criterion of the Householder algorithm is satisfied after 5 iterations :

$$[0, 0] \subseteq \text{residue}(x_4, x_5) \subseteq [-1.44e^{-5}, 1.44e^{-5}]$$

  - Tight enclosure of the iterate :

$$[0.22395, 0.24951] \subseteq x_5 \subseteq [0.22360, 0.25000]$$

  - Functional proof :

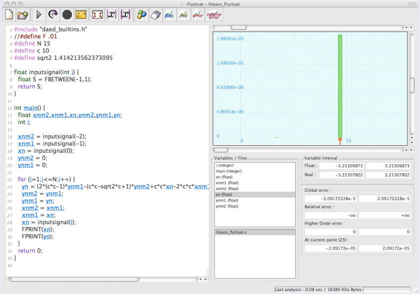
$$[0, 0] \subseteq \text{shouldbezero} \subseteq [-1.49e^{-6}, 1.49e^{-6}]$$

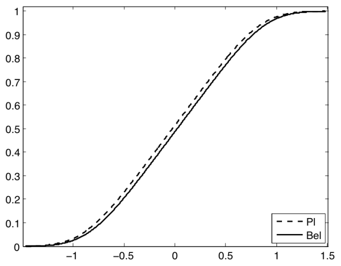
Typical problem

- Some inputs being known set theoretically (non-deterministic inputs) or in probability (probabilistic inputs)
  - e.g. thermal noise in CCD cameras: Gaussian distribution with zero mean and standard deviation varying with temperature according to Nyquist law
- In fact, more generally, inputs may be thought of as given by imprecise probabilities (such as the ones given by probability boxes or P-boxes: pair of upper and lower probabilities)

Example: recursive filter with independent inputs in [-1,1]

Prove that dangerous worst case occur with very low probability

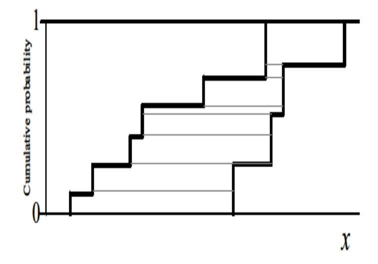




- Deterministic analysis (left): outputs in [-3.25,3.25] (exact)
- Mixed probabilistic/deterministic analysis (right): outputs in [-3.25,3.25], and in [-1,1] with very strong probability (in fact, very close to a Gaussian distribution)

Based on Dempster-Shafer structures (1976)

- Based on a notion of focal elements ($\in F$ - here F is a set of subsets of $\mathbb{R}$):
  - sets of non-deterministic events/values - here sub-intervals of values in [-1,1]
- Weights (positive reals) associated to focal elements ($w : F \to \mathbb{R}^+$)
  - probabilistic information only available on the belonging to the focal elements, not to precise events
  - equivalent to having staircase upper and lower probabilities



(taken from SANDIA 2002-4015)

Our approach

- Encode as much deterministic dependencies as possible

$$\begin{array}{rcll} S_{n+2} & = & 0.7E_{n+2} - 1.3E_{n+1} + 1.1E_n & \text{independent values} \\ & & +1.4S_{n+1} - 0.7S_n & \text{linear dependancy} \end{array}$$

  - use affine arithmetic based abstraction
  - linearization of dependencies
  - representation on a basis of independent noise symbols
- associate a Dempster-Shafer structure to each noise symbol
- technicality: some noise symbols (coming from non-linear terms in particular) have unknown dependencies...
- use of Frechet bounds when dependencies are unknown, easier calculus when variables are known to be independent

P-forms

- Affine forms based on two sets of noise symbols:
  - ε_i independent with each other, created by inputs
  - η_j unknown dependencies with each other and with the ε_i, created by non-linear computation (including branching)
- Together with (imprecise) probabilistic information:
  - Dempster-Shafer structures associated to ε_i: (F^i, w^i) and associated to η_j: (G^j, v^j)

More details:

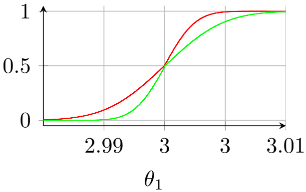
Bouissou et al. Computing 2012 (probabilistic arithmetic) and VSTTE 2013 (abstract domain, correctness with respect to a concrete semantics and join/meet operations)

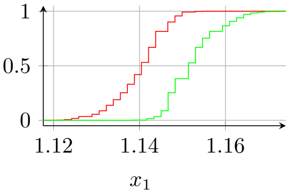
- Example from Enszer, J.A., Lin, Y., Ferson, S., Corliss, G.F., Stadtherr, M.A., "Probability bounds analysis for nonlinear dynamic process models"
- Goal: compute bounds on the solution of the differential equations

$$\dot{x}_1 = \theta_1 x_1(1 - x_2) \quad \dot{x}_2 = \theta_2 x_2(x_1 - 1)$$

with initial values $x_1(0) = 1.2$ and $x_2(0) = 1.1$ and uncertain parameters θ_1, θ_2 given by a normal distribution with mean 3 and 1, resp., but with an unknown standard deviation in the range $[-0.01, 0.01]$

- Results with our probabilistic affine forms:





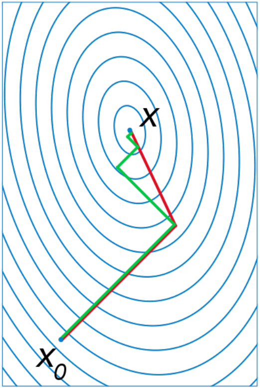
- Application: we can, with high probability, discard some values in the resulting interval. For example, we could show that $P(x_1 \leq 1.13) \leq 0.0552$

Conjugate gradient algorithm: solve $Ax = b$

```
while (norm > epsilon) {          /* residue norm == <gi,gi> */
  evalA(hi,temp);                 /* temp = Ahi */
  rho = scalarproduct(hi,temp);
  norm2 = norm;
  gamma = norm2/rho;              /* gamma = <gi,gi>/<hi,Ahi> */
  multadd(xi,hi,1,gamma,xsi);     /* approx sol xsi = xi + gamma hi */
  multadd(gi,temp,1,-gamma,gsi);  /* residue gsi = gi - gamma temp */
  norm = scalarproduct(gsi,gsi);
  beta = norm/norm2;              /* beta = <gsi,gsi>/<xi,xi> */
  multadd(gsi,hi,1,beta,hsi);     /* direction hsi = gsi + beta hi */
  for (j=0;j<N;j++) {
    xi[j] = xsi[j];
    gi[j] = gsi[j];
    hi[j] = hsi[j];
  }
}
```

In real numbers: for A symmetric positive definite ($\forall x, < x, Ax > \geq 0$)

- the successive directions *hsi* are conjugate ($< Ah_i, h_{i+1} >= 0$),
- the exact solution (in real numbers) is found in at most N iterates (N the size of matrix A).

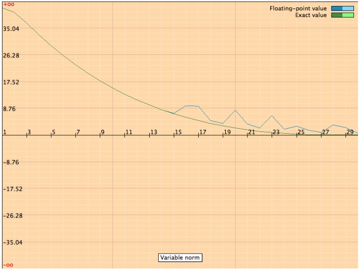


In real numbers: for A symmetric positive definite ($\forall x,\ < x, Ax >\geq 0$)

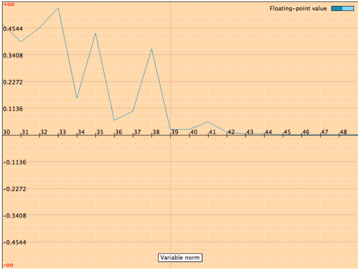
- the successive directions *hsi* are conjugate ($< Ah_i, h_{i+1} >=0$),
- the exact solution (in real numbers) is found in at most N iterates (N the size of matrix A).

Matrix A is now Strakos matrix in dimension 30

- Condition number around 1000
- Convergence in 30 iterations in real numbers but more difficult in float

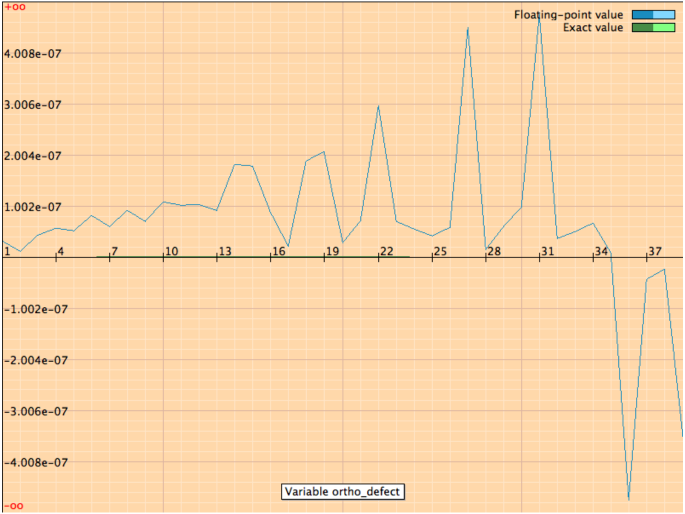


Float and real value of the norm



Norm in float for iterates > 30

Orthogonality defect



Orthogonality defect $\frac{<Ah_i, h_{i+1}>}{\|Ah_i\|\|h_{i+1}\|}$