Static Analysis by Abstract Interpretation of Numerical Programs and Systems - Fluctuat

Eric Goubault and Sylvie Putot

Cosynus, LIX, Ecole Polytechnique (ex MeASI, CEA LIST)

Journée Aristote, Accréditation des résultats de la recherche, 19/12/2014

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/convergence) of finite precision computations
  - method error

Context: safety-critical programs

Typically flight control or industrial installation control

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

Computer-aided approaches to the problem of uncertainties and roundoff

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, etc.)
- 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 sets or local 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

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

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

Example: interpretation in (products of) intervals

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)

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

Affine forms

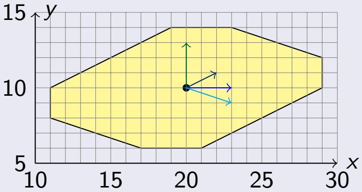
- Affine form for variable x:

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

  where the ε_i are symbolic variables (*noise symbols*), with value in $[-1, 1]$.
- Sharing ε_i between variables expresses implicit dependency
- Affine arithmetic: close to Taylor Methods of low degree

Geometric concretization as zonotopes (center symmetric polytopes)

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

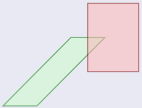


Huge litterature - (dual) generator representation of a polytope!

Set operations on zonotopes

Test Interpretation

- Intersection of zonotopes (or zonotopes with guards) are not zonotopes!



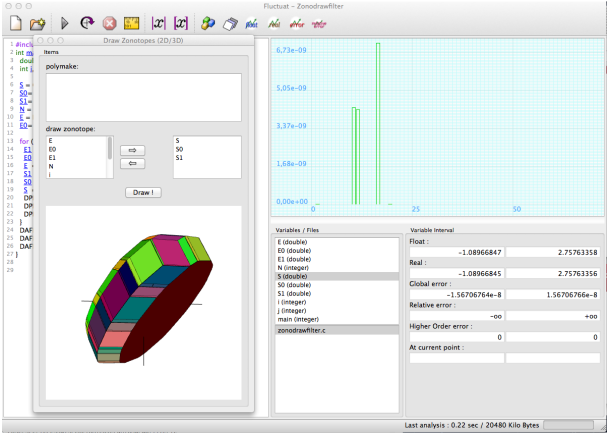
- Need a tight over-approximation of this intersection
- Solution: affine forms + constraints on noise symbols

Join Operation

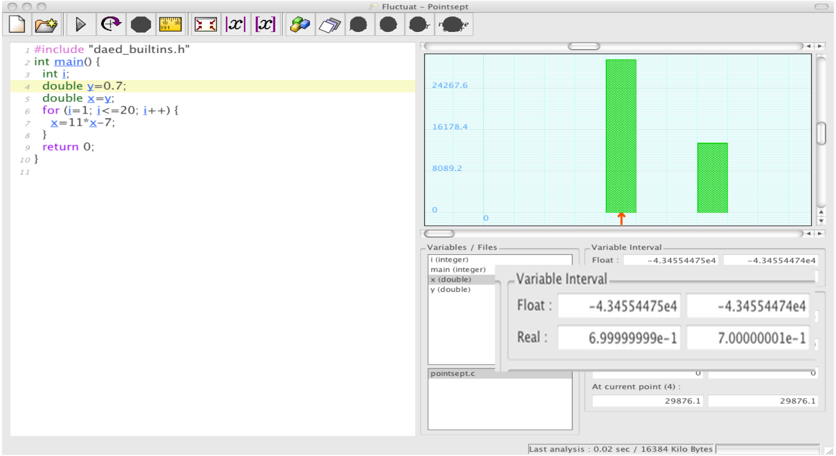
- No least upper bound in general.
- Compute a (minimal) upper bound

- Convergence of Kleene iteration with zonotopes linked to convergence of numerical scheme
- Ex: for linear recursive filters, if numerical scheme is bounded, there is a Kleene-like iteration that can prove it (for unbounded time horizon)

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



Fluctuat: computes rounding errors and propagates errors and uncertainties



- Relying on affine forms both for real value and error terms;
- with two sets of constraints on the noise symbols, resp. corresponding to real and finite control flows

Householder scheme for square root

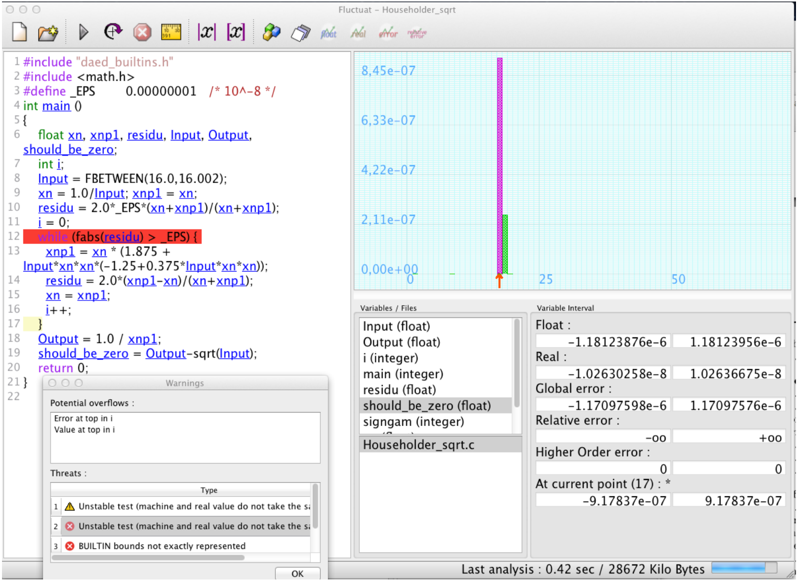
Householder

Control flow difficulties

Unstable tests: when control flow for real number and floating-point computations can be different

- Classical (straight line programs) error analyses are unsound in these cases
- When considering large sets of executions, most tests are possibly unstable (just issuing a warning is not practical)
- Bound the discontinuity error between the two branches under conditions of unstable tests (mix constraint solving / affine arithmetic)
- Robustness analysis of implementations

Householder algorithm for square root



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

```
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).

Are we done?

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

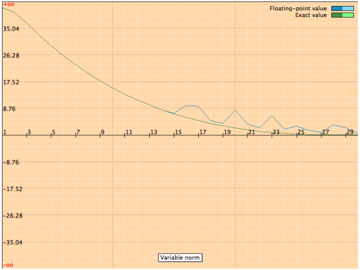
Conjugate gradient algorithm: solve $Ax = b$

```
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];
  }
}
```

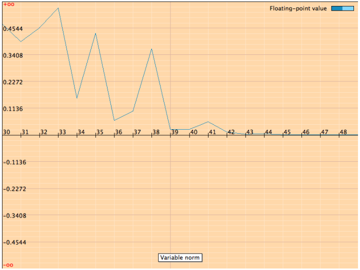
In real numbers: for A symmetric positive definite ($\forall x, < x, Ax > \geq 0$)

- 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).

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

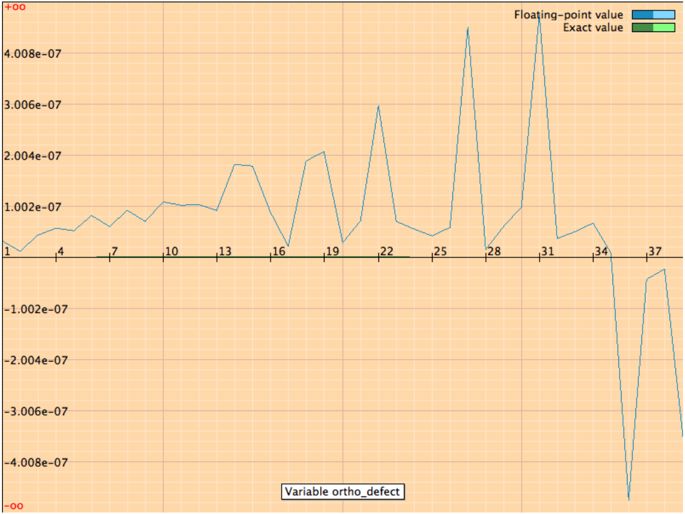


Float and real value of the norm



Norm in float for iterates > 30

Orthogonality defect



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

This is still a long term goal...

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 be extended to deal with other or more refined properties: finite precision semantics, polynomial abstractions (ellipsoids), under-approximations (using Kaucher arithmetic), hybrid systems analysis, probabilistic systems (mix with pboxes / dempster shafer structures) etc.

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