Vers l'analyse statique de programmes numériques

Sylvie Putot

Laboratoire de Modélisation et Analyse de Systèmes en Interaction, CEA LIST

Journées du GDR et réseau Calcul, 9-10 novembre 2010

Motivations

- Validate numerical programs
  - Validate an implementation in finite precision
    - prove that the program computes something close to what is expected:
    - accuracy of results
    - behaviour of the program (control flow, number of iterations)
  - But also validate the algorithm
    - bound when possible the method/approximation error
- 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

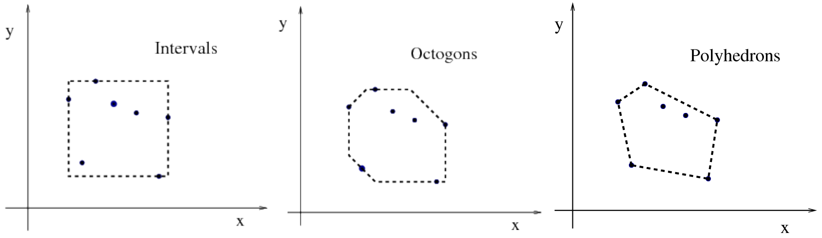
- Introduction to static analysis
- Static analysis of numerical program
- Applications - the Fluctuat static analyzer

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 ordered sets (abstract lattice)



- Interpret computations in this lattice

Invariants

- Invariants allow to conclude about the *safety* (for instance absence of run-time errors) of programs
- Program seen as a large discrete dynamical system
  - 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)
  - these equations are constructed from the dynamics of the program (using labelled directed graphs)
  - safe approximation in the abstract lattice
- The sets of possible values of variables at control points are invariants, computed as the least fixpoint of the system
  - usually solved iteratively (Kleene)
  - convergence acceleration to terminate in finite time

Example in intervals

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

$$X = F(X):$$

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

$(I(\mathbb{R}), \subseteq)$ is a complete lattice and F is monotonic so there exists a least fixpoint, it is the limit of the Kleene iteration $X^0 = \bot$, $X^1 = F(X^0)$, ..., $X^{k+1} = X^k \cup F(X^k)$

Kleene iterations ($F(\perp)$)

$$
\begin{aligned}
x_0^0 &= \top \\
x_1^0 &= [-100, 50] \\
x_2^0 &= [-100, 50] \\
x_3^0 &=]-\infty, 99] \cap ([-100, 50]) \\
&= [-100, 50] \\
x_4^0 &= [-100, 50] + [1, 1] \\
&= [-99, 51] \\
x_5^0 &= [100, +\infty[\cap ([-100, 50]) \\
&= \perp
\end{aligned}
$$

Kleene iterations ($F^2(\perp)$)

$$\begin{array}{rcl}
x_0^1 & = & \top \\
x_1^1 & = & [-100, 50] \\
x_2^1 & = & [-100, 51] \\
x_3^1 & = &]-\infty, 99] \cap ([-100, 51]) \\
 & = & [-100, 51] \\
x_4^1 & = & [-100, 51] + [1, 1] \\
 & = & [-99, 52] \\
x_5^1 & = & [100, +\infty[\cap ([-100, 51]) \\
 & = & \perp
\end{array}$$

Kleene iterations ($F^{50}(\perp)$)

$$\begin{array}{rcl}
x_0^{50} & = & \top \\
x_1^{50} & = & [-100, 50] \\
x_2^{50} & = & [-100, 100] \\
x_3^{50} & = &]-\infty, 99] \cap ([-100, 100]) \\
 & = & [-100, 99] \\
x_4^{50} & = & [-100, 99] + [1, 1] \\
 & = & [-99, 100] \\
x_5^{50} & = & [100, +\infty[\cap ([-100, 100]) \\
 & = & [-100, 100]
\end{array}$$

Outline of the talk

- Introduction to static analysis
- Static analysis of numerical program
  - modelling finite precision
  - abstraction based on affine arithmetic
- Applications - the Fluctuat static analyzer

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:
    $\frac{1}{10} = 0.000110011001100110011001100\cdots$ (binary)
    $\Rightarrow \mathit{float}(\frac{1}{10}) = 0.100000001490116119\cdots$ (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

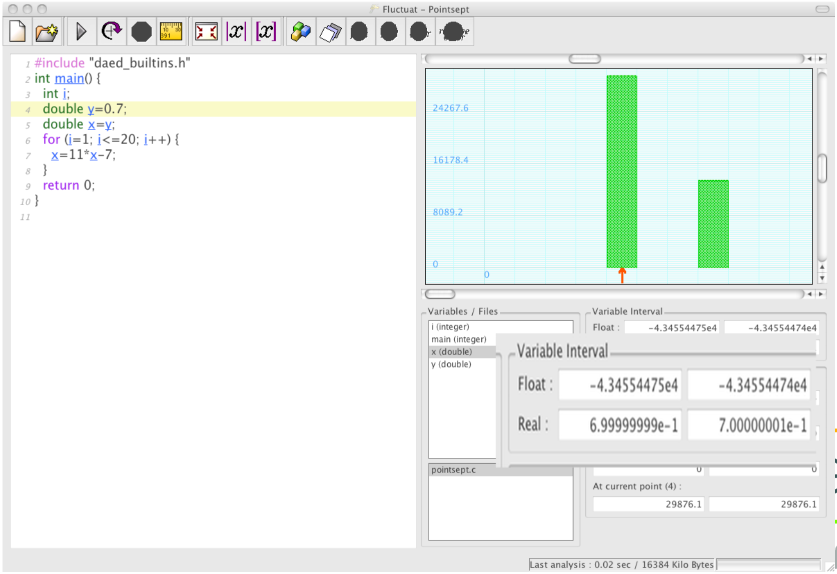
Our model for the analysis of numerical computations

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

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

$\Rightarrow$ Then abstraction for each term (real value and errors)

Example



```
#include "daed_builtins.h"
int main() {
  int i;
  double y=0.7;
  double x=y;
  for (i=1; i<=20; i++) {
    x=11*x-7;
  }
  return 0;
}
```

Variable Interval

Float : -4.34554475e4 -4.34554474e4

Real : 6.99999999e-1 7.00000001e-1

At current point (4) :

29876.1 29876.1

Last analysis : 0.02 sec / 16384 Kilo Bytes

Abstract model

- Elementary rounding error when rounding r^x to $\uparrow_\circ r^x$: there exist $\delta_r > 0$ and $\delta_a > 0$

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

- For each variable x, a triplet (r^x, f^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, i.e. of initial or rounding errors and their propagation in computations
- Abstraction of r^x and e^x using guaranteed ensemblist methods:
  - the simplest: intervals
  - affine forms (zonotopes)
  - ellipsoids in the future, etc.

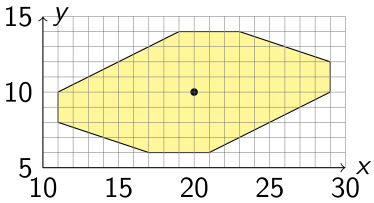
Abstraction based on Affine Arithmetic (Stolfi 93)

- 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 ε_i are independent symbolic variables with unknown value in $[-1, 1]$.
- Sharing ε_i between variables expresses *implicit dependency*: concretization as a zonotope

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



Abstract domain based on affine arithmetic

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

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

- *Non linear operations* : approximate linear form (Taylor expansion), new noise term for the approximation error

Set-theoretical operations?

- Partial order: to compute a fixpoint on an ordered structure
- Intersection: to interpret conditionals
- Join: to collect abstract values from different control flows

Zonotopes for functional abstraction

- Add "perturbation" part to the "central" affine sets, e.g.:

$$\begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = \begin{pmatrix} 10 & -4\varepsilon_1 & & +2\varepsilon_3 & +3\varepsilon_4 & \boxed{\begin{matrix} +1\eta_1 & -3\eta_2 \\ & +1\eta_2 \end{matrix}} \\ 5 & -2\varepsilon_1 & +1\varepsilon_2 & & -1\varepsilon_4 & \end{pmatrix}$$

$$= {}^tC^t(1, \varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4) + {}^tP^t(\eta_1, \eta_2)$$

- The central part links the current values of the program variables to the initial values of the input variables (linear functional)
  - the ε_i are fixed
- The perturbation part encodes the uncertainty in the description of values of program variables due to the non-linear computations (multiplication, join etc.)
  - we allow more freedom on the η_i

Functional order relation

- For X and Y, two perturbed affine sets over p variables, $X \leq Y$ iff

$$\sup_{u \in \mathbb{R}^p} \left(\|(C^Y - C^X)u\| + \|P^X u\| - \|P^Y u\| \right) \leq 0$$

☛ Implies the geometric ordering

☛ Equivalent to the geometric ordering on zonotopes enclosing current values of variables + slack variables representing their initial values

Minimal upper bounds

Let $\sqsubseteq$ be a partial order on a set X. We say that z is *a mub* of two elements x, y of X if and only if

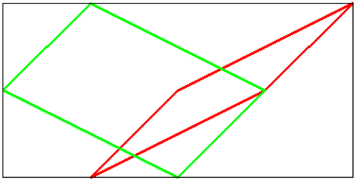
- z is an upper bound of x and y, i.e. $x \sqsubseteq z$ and $y \sqsubseteq z$,
- for all z' upper bound of x and y, $z' \sqsubseteq z$ implies $z = z'$.

In our case:

- no least upper bound in general, but several minimal upper bounds
- we define a join operator, which is an upper bound, and a minimal upper bound in some cases

Example computation of a (m)ub

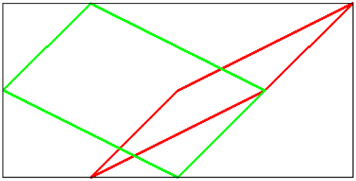
$$\begin{aligned}\hat{x} &= 3 + \varepsilon_1 + 2\varepsilon_2\\ \hat{y} &= 1 - 2\varepsilon_1 + \varepsilon_2\\ \hat{u} &= \varepsilon_1 + \varepsilon_2\end{aligned}$$



$\hat{x}$ et $\hat{y}$ functions of $\hat{u}$

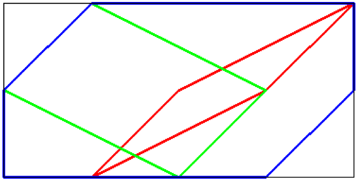
Example computation of a (m)ub

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



$\hat{x}$ et $\hat{y}$ functions of $\hat{u}$

$$\hat{z} = \hat{x} \cup \hat{y} = 2 + \varepsilon_2 + 3\varepsilon_U$$



$\hat{x}$, $\hat{y}$ et $\hat{z}$ functions of $\hat{u}$

$$\gamma(\hat{z}) = [-2, 6] = \gamma(\hat{x}) \cup \gamma(\hat{y})$$

Join over affine forms

We define $z = x \cup y$ by

$$\begin{cases} \alpha_0^z = mid([\hat{x}] \cup [\hat{y}]) \\ \alpha_i^z = \underset{\alpha_i^x \wedge \alpha_i^y \leq \alpha \leq \alpha_i^x \vee \alpha_i^y}{\operatorname{argmin}} |\alpha|, \ \forall i \geq 1 \\ \beta^z = \sup \gamma(\hat{x}) \cup \gamma(\hat{y}) - \alpha_0^z - \|z\|_1 \end{cases}$$

$$\underset{u \wedge v \leq \alpha \leq u \vee v}{\operatorname{argmin}} |\alpha| = \{\alpha \in [u \wedge v, u \vee v], |\alpha| \text{ minimal}\}$$

Intuitively, α_i^z expresses the common dependency to symbol ε_i, and the remainder is associated to a new noise symbol ε_U

- efficient (linear in the number of symbols, and eliminates part of the symbols)
- range of values taken by the union is the union of the ranges

Intersections: main idea, informally

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

- Affine forms before tests: $x = 5 + \varepsilon_1$, $y = 10 + 2\varepsilon_1$
- 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.
- Here, in intervals, deduce that $y = 5 + \varepsilon_1$ with $\varepsilon_1 \in [0, 1]$ in the `then` branch, implying $y \in [0, 10]$ at the end of the program

Outline of the talk

- Introduction to static analysis
- Static analysis of numerical program
- Applications - the Fluctuat static analyzer

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
- But also, in some cases, weak functional proof of algorithms
- 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

Back to the Householder scheme

Householder

Householder scheme: summary

- Bounds on the values of the number of iterates:
  - proves the good behaviour of the real algorithm and finite precision implementation
- Bounds on the real value of should_be_zero:
  - proves the method error is small
- Bounds on the error of x_n and should_be_zero:
  - proves the finite precision implementation is accurate,
  - should be compared to the method error, to find the good match between method and implementation errors

Current research

- Applications to numeric codes such as:
  - conjugate gradient/Lanczos algorithms (first experiments made on `BCSSTK01` and others, also on sets of symmetric definite positive matrices)
  - integration schemes (the effect of step choice wrt precision in particular)
  - finite difference schemes, finite element schemes
- Improved relational domains
- Also, computation of under-approximations → show the quality of results (plus scenarios for "extreme cases")
- Improvement of the resolution of semantic equations by policy iteration techniques: better precision and faster+incremental analysis
- Combination of deterministic and probabilistic methods for less pessimistic estimation of the errors of computation

References

- Abstract domains based on affine forms for the computation of invariants on numerical programs:
  - Static Analysis of Numerical Algorithms, *SAS 2006 (Static Analysis Symposium)*
  - Perturbed affine arithmetic for invariant computation in numerical program analysis, *arXiv:0807.2961, july 2008*
  - The Zonotope Abstract Domain Taylor1+, *CAV 2009 (Computer Aided Verification)* and its extension in *CAV 2010*
  - A Zonotopic Framework for Functional Abstractions, *arXiv:0910.1763, october 2009* - better version submitted 2010
  - Static Analysis of Finite Precision Computations, *VMCAI 2011 (Verification, Model-Checking and Abstract Interpretation)*

References

- Implementation and use of FLUCTUAT and industrial case studies
  - Static Analysis of the Accuracy in Control Systems: Principles and Experiments (with IRSN - Institut de Radioprotection et de Sécurité Nucléaire - and Hispano Suiza), *FMICS 2007 (Formal Methods for Industrial Critical Systems)*
  - 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/`

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: Sylvie.Putot@cea.fr