Static Analysis by Abstract Interpretation
From numerical programs to systems

Eric Goubault and Sylvie Putot
Cosynus team, LIX, Ecole Polytechnique

24th of november, 2015
A long story made short

```c
#include "daed_builtins.h"
#include <math.h>
#define _EPS 0.00000001 /* 10^-8 */

int main ()
{
    float xn, xnp1, residu, Input, Output,
    should_be_zero;
    int i;
    Input = FBETWEEN(16.0,16.002);
    xn = 1.0/Input; xnp1 = xn;
    residu = 2.0*_EPS*(xn+xnp1)/(xn+xnp1);
    i = 0;
    while (fabs(residu) > _EPS) {
        xnp1 = xn * (1.875 +
        Input*xn*xn*(-1.25+0.375*Input*xn*xn));
        residu = 2.0*(xnp1-xn)/(xn+xnp1);
        xn = xnp1;
        i++;
    }
    Output = 1.0 / xnp1;
    should_be_zero = Output-sqrt(Input);
```
A long story made short

```c
#include "daed_builtins.h"
#include <math.h>
define _EPS 0.00000001 /* 10^-8 */
int main ()
{
    float xn, xnp1, residu, Input, Output,
    should_be_zero;
    int i;
    Input = FBETWEEN(16.0,16.002);
    xn = 1.0/Input; xnp1 = xn;
    residu = 2.0*_EPS*(xn+xnp1)/(xn+xnp1);
    i = 0;
    while (fabs(residu) > _EPS) {
        xnp1 = xn * (1.875 +
        Input*xn*xn*(-1.25+0.375*Input*xn*xn));
        residu = 2.0*(xnp1-xn)/(xn+xnp1);
        xn = xnp1;
        i++;}
Output = 1.0 / xnp1;
should_be_zero = Output-sqrt(Input);
```
From programs...to cyberphysical systems

A long story made short

(Embedded France)
Invariants, reachability, variants, abstractions...

**Program**

Control points:
level of granularity in “time”

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

Abstraction: here, intervals, granularity in “space”

```
\[
\begin{align*}
  x_1 & = [-100, 50] \\
  x_2 & = x_1 \cup x_4 \\
  x_3 & = ] - \infty, 99] \cap x_2 \\
  x_4 & = x_3 + [1, 1] \\
  x_5 & = [100, +\infty[ \cap x_2
\end{align*}
\]
```
Invariants, reachability, variants, abstractions...

Program

Control points:
level of granularity in “time”

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

Abstraction: here, intervals, granularity in “space”

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

- Invariants: at [2], $-100 \leq x \leq 100$
- Variants: at [2], $100 - x$ is decreasing and positive (termination)
Differential equations

\[
\begin{align*}
\frac{dx}{dt} &= -x + 2x^2y \\
\frac{dy}{dt} &= -y
\end{align*}
\]

Lyapunov function \( R(x, y) = \frac{y^2}{2} + \frac{x^2}{2(1-xy)} \) (Hahn, ACC 2014):
- Invariant: e.g. for \( x_0y_0 < 1 \), \( \{(x, y) \mid R(x, y) \leq R(x_0, y_0)\} \) is invariant
- Variant: for \( x_0y_0 < 1 \), \( R \) is decreasing and positive on all trajectories (stability)
Variant: convergence towards an escape state (quaternion equal to (1, 1, 1, 1))

```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);
    }
}
```

T1 gets a and b before T2 => \( a=2 \) and \( b=4 \)

T2 gets b and a before T1 => \( a=2 \) and \( b=3 \)

Each of T1 and T2 gets a resource
=> Deadlock with \( a=2 \) and \( b=1 \)

Topological invariants:

- 2 possible schedules and a deadlock (and an unreachable region)
- implies also a numerical invariant \( (a = 2, \ b = 3 \) or \( a = 2, \ b = 4 \) or deadlock)

Part I. Numerical Programs
Example: Householder scheme for square root approx
What is “correctness” for numerical computations?

- No run-time error (division by 0, overflow, etc), see Astrée, Frama C, Polyspace... 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)
Set-based methods and Abstract Interpretation

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]
```

\[ X = F(x) \]

\[
\begin{align*}
  x_1 &= [-100, 50] \\
  x_2 &= x_1 \cup x_4 \\
  x_3 &= ] - \infty, 99[ \cap x_2 \\
  x_4 &= x_3 + 1 \\
  x_5 &= [100, +\infty[ \cap x_2
\end{align*}
\]
Set-based methods and Abstract Interpretation

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
    \[
    X^0 = \bot, X^1 = F(X^0), \ldots, X^{k+1} = X^k \cup F(X^k)
    \]
- 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
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
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 \( \varepsilon_i \) are symbolic variables (*noise symbols*), with value in \([-1, 1]\).

- Sharing \( \varepsilon_i \) 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)**

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

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

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

- **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

\[
\hat{x} \times \hat{y} = x_0y_0 + \sum_{i=0}^{n} (x_0y_i + x_iy_0)\varepsilon_i + \left( \sum_{1 \leq i \neq j \leq n} |x_iy_j| \right) \varepsilon_{n+1}
\]

(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)
A simple example: functional interpretation

```plaintext
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

$$y = 32.5 + 50\varepsilon_1 + 12.5\eta_1$$
A simple example: functional interpretation

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

Abstraction of $x$: $x = 5 + 5\varepsilon_1$

Abstraction of function $x \rightarrow y = x^2 - x$ as

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

$$ = -17.5 + 10x + 12.5\eta_1 $$
Set operations on affine sets / zonotopes: meet

Reminder

\[ \text{int } x = [-100, 50]; \]
\[ \text{while } (x < 100) \]
\[ x = x + 1; \]
\[ X = F(x) \]

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

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)
Intersection of zonotopes are not zonotopes!

Example

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

- Affine forms before tests: $x = 5 + 5\epsilon_1$, $y = 10 + 10\epsilon_1$
- In the if branch $\epsilon_1 \geq 0$: 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”

\[
z_i = \arg\min_{x_i \land y_i \leq r \leq x_i \lor y_i} |r|, \ \forall i \geq 1
\]
Join operator

\[
\left(\begin{array}{l}
\hat{x} = 3 + \varepsilon_1 + 2\varepsilon_2 \\
\hat{u} = 0 + \varepsilon_1 + \varepsilon_2
\end{array}\right) \cup \left(\begin{array}{l}
\hat{y} = 1 - 2\varepsilon_1 + \varepsilon_2 \\
\hat{u} = 0 + \varepsilon_1 + \varepsilon_2
\end{array}\right) = \left(\begin{array}{l}
\hat{x} \cup \hat{y} = 2 + \varepsilon_2 + 3\eta_1 \\
\hat{u} = 0 + \varepsilon_1 + \varepsilon_2
\end{array}\right)
\]

Construction (low complexity!): \( O(n \times p) \)

- Keep “minimal common dependencies”

\[
z_i = \arg\min_{x_i \wedge y_i \leq r \leq x_i \vee y_i} \ |r|, \ \forall i \geq 1
\]

- 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
Convergence results: from concrete to abstract

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

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

- Concrete scheme has **bounded outputs** iff zeros of \( x^n - \sum_{i=0}^{n-1} a_{i+1} x^i \) have modulus strictly 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

\[ S_{n+2} = 0.7E_{n+2} - 1.3E_{n+1} + 1.1E_n + 1.4S_{n+1} - 0.7S_n \]
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
  \[
  \frac{\log 2}{\log \sup_{\lambda \text{eigenvalue}} |\lambda|}
  \]
  (here spectral radius of 0.84, but \( q \sim 15 \) 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, EMSOFT 2015
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)

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

\[
\begin{align*}
f^x & = 0.1 + 1.49e^{-9} \quad [1] \\
f^y & = 0.5 \\
f^z & = 0.6 + 1.49e^{-9} \quad [1] + 2.23e^{-8} \quad [3] \\
f^t & = 0.06 + 1.04e^{-9} \quad [1] + 2.23e^{-9} \quad [3] - 8.94e^{-10} \quad [4] - 3.55e^{-17} \quad [ho]
\end{align*}
\]
```c
#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;
}
```
Abstract value

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

\[
 f^x = (\alpha_0^x + \bigoplus_i \alpha_i^x e_i^r) + (e_0^x + \bigoplus_i e_i^x e_i^e)
\]

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

24th of November, 2015

Static Analysis by Abstract Interpretation From numerical programs to systems
Back to the Householder scheme
#include "daed_builtins.h"
#include <math.h>
define _EPS 0.00000001 /* 10^-8 */

int main()
{
    float x0, x001, residu, Input, Output,
         should_be_zero;
    int i;
    Input = FBETWEEN(16.0, 16.002);
    x0 = 1.0 / Input; x001 = x0;
    residu = 2.0*_EPS*(x0 + x001)/(x0 + x001);
    i = 0;
    while (fabs(residu) > _EPS){
        x001 = x0 * (1.875 +
                   Input*x0*x0*1.25 + 3.75*Input*x0*x0);
        residu = 2.0*(x001-x0)/(x0 + x001);
        x0 = x001;
        i++;
    }
    Output = 1.0 / x001;
    should_be_zero = Output - sqrt(Input);
    return 0;
}
Filters
To Systems!
**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);
    }
}
```

- 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).
Extensions of affine sets

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 (\( \varepsilon_i = [-1, 1] \) or \( \varepsilon^*_i = [1, -1] \)), Kaucher arithmetic extends classical interval arithmetic (SAS 2007, HSCC 2014 with M. Kieffer)
- Noise symbols \( \varepsilon_i \) coding sets of probability distributions:
  - **probabilistic affine forms**: \( \varepsilon_i \) take values in probability boxes (Computing 2012, and VSTTE 2013, with O. Bouissou, J. Goubault-Larrecq)
  - plus use of Chernoff-Hoeffding bounds (with S. Sankaranarayanan) and some moments computations
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)
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.)
- Applications to the primary flight computers of the A380, A350; to critical parts of the safety mechanisms of nuclear plants, of the ATV etc.
- see e.g. FMICS 2007, 2009, DASIA 2009

Still...

- Rather simple numerical computations: linear recursive filters, linear control, mathematical libraries (at the exception of Astrium’s ATV)
- What about cyber-physical systems, i.e. distributed control programs (some other talk!)?
- What about simulation programs such as finite element methods etc. (some other talk too)?
- We are heavily investing in abstractions for ODEs, DAEs etc.:
  - “Symbolic methods” for deriving weak Lyapunov functions of ODEs, DAEs, not only polynomial, but rational, transcendental...(ideas from computer algebra - see ACC 2014 with S. Sankaranarayanan)
  - “Geometric methods” for finding out regions in which a (positive) invariant exists, and even controllability of switched systems...(ideas from Conley index theory - see RP 2015 with M. Mrozek, L. Eribourg, S. Mohamed)
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.
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...!