DE LA RECHERCHE À L'INDUSTRIE

Static Analysis of Numerical Programs and Systems

Sylvie Putot

joint work with O. Bouissou, K. Ghorbal, E. Goubault, T. Le Gall, K. Tekkal, F. Védrine

Laboratoire LMEASI, CEA LIST | Digicosme Spring School, 22-26 April, 2013

Automatic validation of numerical programs and systems

- No run-time error (division by 0, overflow, out-of-bounds array access, etc)
- Validate algorithms: bound when possible the method error
  - Check functional properties in real-number semantics
- Validate finite precision implementations: prove the program computes something close to expected (in real numbers)
  - Accuracy of results
  - Behaviour of the program (control flow, number of iterations)
- Context: safety-critical programs
  - Typically flight control or industrial installation control (signal processing, instrumentation software)
  - Sound and automatic methods
    - Guaranteed, that prove good behaviour or else give counter-examples
    - Automatic, for a program and sets of (possibly uncertain) inputs/parameters

Abstract interpretation based static analysis

A simple example

```c
/* float-error.c */
int main () {
    float x, y, z, r;
    x = 1.0e11 - 1;
    y = 1.0e11 + 1;
    z = x - y;
    r = 1/z;
    printf("%f %f\n", z, r);
}
```

A simple example

```c
/* float-error.c */
int main () {
    float x, y, z, r;
    x = 1.0e11 - 1;
    y = 1.0e11 + 1;
    z = x - y;
    r = 1/z;
    printf("%f %f\n", z, r);
}
```

gcc float-error.c
./a.out
> 0.000000 inf

Example: Householder scheme for square root computation

Example: Householder scheme for square root computation

- Can we signal potential problems and their origin ?
- Can we prove:
  - that the algorithm seen in real numbers computes something close to what is expected (here, the square root of the input)?
  - that it does so also in finite precision?
  - that the finite precision behaviour (control flow) of the scheme is satisfying?

Outline

1. Abstract interpretation based static analysis
   - The foundations
2. From affine arithmetic to zonotopic abstract domains
   - The full construction and study of a class of numerical abstract domains, possible extensions
3. Analysis of floating-point computations, case studies
   - Extensions of the zonotopic domains for finite precision analysis, specific problems such as unstable tests, the Fluctuat static analyzer
4. Some variations of the zonotopic abstract domains

I. Abstract interpretation based static analysis

Static Analysis of Programs

The goals of static analysis: prove properties on the program analyzed

- fully automatically
- without executing the program, for (possibly infinite) sets of inputs and parameters

Some properties

- invariant properties (true on all trajectories - for all possible inputs or parameters).
  Example : bounds on values of variables, absence of run-time errors
- liveness properties (that become true at some moment on one trajectory).
  Examples : state reachability, termination

Two main influences

- Formal meaning to data-flow analyses in compiler optimizations (constant propagation, use-def analysis, parallelisation - Kildall's lattice 73, Karr's lattice 76):
  Replace:

```
int f(int j)
{
  int i,j;
  i = 42;
  while (j<100)
    j = 2*i+1+2*
        j;
  return j;
}
```

by...

```
int f(int j)
{
  int i;
  i = 42;
  while (j<100)
    j = 85+2*j;
  return j;
}
```

and even...

```
int f(int j)
{
  while (j<100)
    j = 85+2*j;
  return j;
}
```

Two main influences

- Formal meaning to compiler optimizations
- Program proof using Hoare logics

Seminal papers

- Patrick Cousot, Radhia Cousot, “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”, POPL 1977
- Patrick Cousot, Nicolas Halbwachs, “Automatic Discovery of Linear Restraints Among Variables of a Program”, POPL 1978
- Patrick Cousot, Radhia Cousot, “Systematic Design of Program Analysis Frameworks”, POPL 1979

Example: can we prove that there is no runtime error in this program?

```
fft(a,n)
 { cplx b[n/2], c[n/2];
   if (n > 2)
    { for (i=0;i<n; i=i+2)
        { b[i/2]=a[i];
          c[i/2]=a[i+1]; }
      fft(b,n/2);
      fft(c,n/2);
      for (i=0;i<n; i=i+1)
       a[i]=F1(n)*b[...]+F2(n)*c[...];}
   else
      a[0]=g*a[0]+d*a[1];
      a[1]=a[0]-2*d*a[1]; } }
```

Example: can we prove that there is no runtime error in this program?

```
fft(a,n)
 { cplx b[n/2], c[n/2];
   if (n > 2)
    { for (i=0;i<n; i=i+2)
       { b[i/2]=a[i];
         c[i/2]=a[i+1]; }
      fft(b,n/2);
      fft(c,n/2);
      for (i=0;i<n; i=i+1)
       a[i]=F1(n)*b[...]+F2(n)*c[...];}
   else
      a[0]=g*a[0]+d*a[1];
      a[1]=a[0]-2*d*a[1];  } }
```

No!

- Erroneous executions when starting with `n` not a power of 2
- Example: `n`=3:
  - `i`=2, interpret `c[i/2]=a[i+1]`
  - but `i`+1 is 3, out of arrays bounds! (`a[0..2]`)

Example: can we prove that there is no runtime error in this program?

```
fft(a,n)
 { cplx b[n/2], c[n/2];
   if (n > 2)
    { for (i=0;i<n; i=i+2)
       { b[i/2]=a[i];
         c[i/2]=a[i+1]; }
      fft(b,n/2);
      fft(c,n/2);
      for (i=0;i<n; i=i+1)
       a[i]=F1(n)*b[...]+F2(n)*c[...];}
   else
      a[0]=g*a[0]+d*a[1];
      a[1]=a[0]-2*d*a[1]; } }
```

No!

- Erroneous executions when starting with `n` not a power of 2

But yes when `n` is a power of 2

Example: looking at proof statements

```
fft(a,n)
// a.length=n ∧ ∃k > 0 n=2^k
 { cplx b[n/2], c[n/2];
   // a.length=n ∧ ∃k > 0 n=2^k ∧ b.length=n/2 ∧ c.length=n/2
   if (n > 2)
    { for (i=0;i<n; i=i+2)
        {
// a.length=n ∧ ∃k > 0 n=2^k ∧ b.length=c.length=n/2 ∧ i≥0 ∧ i<n ∧ ∃j ≥ 0 i=2j
          b[i/2]=a[i];
          // i+1<n
          c[i/2]=a[i+1]; }
      fft(b,n/2);
      fft(c,n/2);
      for (i=0;i<n; i=i+1)
       a[i]=F1(n)*b[...]+F2(n)*c[...];}
   else
    // a.length=2
      a[0]=g*a[0]+d*a[1];
      a[1]=a[0]-2*d*a[1];  } }
```

Example: can we automatize a proof of partial correctness?

- When trying to prove "simple" statements about programs, can we automatize the synthesis of (strong enough) loop invariants?
- Here we want to be able to bound the indexes for array dereferences to prove absence of array out of bounds accesses
- We see we only need linear inequalities between variables (`i` and `n` in particular) and parity of `i` and of `n` at all recursive calls

Proofs and abstract interpretation

Example: can we automatize a proof of partial correctness?

- When trying to prove "simple" statements about programs, can we automatize the synthesis of (strong enough) loop invariants?

Main idea: carefully choose some particular predicates

- Abstract all first-order predicates into these chosen predicates:
- So that proofs on these abstract predicates become tractable
- And abstract predicates are strong enough to prove some properties of interest (e.g. absence of runtime errors)

Example: can we automatize a proof of partial correctness?

- When trying to prove "simple" statements about programs, can we automatize the synthesis of (strong enough) loop invariants?

Main idea: carefully choose some particular predicates

- Abstract all first-order predicates into these chosen predicates:
  Here: necessary predicates are of the form $(v_i - v_j \leq c_{ij})$ (octagons) and $(v_i$ is even) and $(v_i$ is a power of 2)
- So that proofs on these abstract predicates become tractable
- And abstract predicates are strong enough to prove some properties of interest (e.g. absence of runtime errors)

Example: can we automatize a proof of partial correctness?

- When trying to prove "simple" statements about programs, can we automatize the synthesis of (strong enough) loop invariants?

Main idea: carefully choose some particular predicates

- Abstract all first-order predicates into these chosen predicates:
  Here: necessary predicates are of the form $(v_i - v_j \leq c_{ij})$ (octagons) and (v_i is even) and (v_i is a power of 2)
- So that proofs on these abstract predicates become tractable
- And abstract predicates are strong enough to prove some properties of interest (e.g. absence of runtime errors)
  Here:
  - at `c[i/2]=a[i+1]`: (`i` is even) and (`i-n`$\leq$ -1) and (`n` is a power of 2) imply (`i`+1 is odd) and (`i`+1 $\leq$ `n`) and (`n` is even) imply (`i`+1 $<$ `n`)
  - at `else`: `n`=2 since (`n` is a power of 2) and (`n` $\leq$ 2) (`else` condition) and (`a.length`=`n`) (part of the loop invariant) imply `a.length`=`n`=2

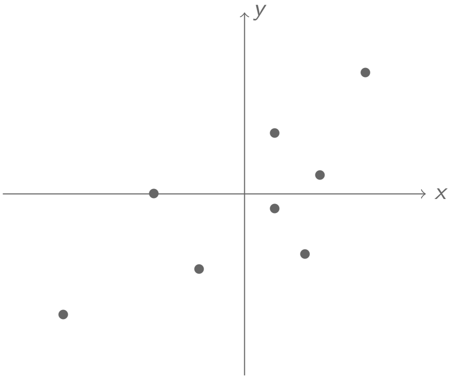
Numerical abstract domains

- These inferences can be made into an algorithm (interpreting program statements by transfer functions in so called abstract domains)
- We will focus in the sequel on numerical abstract domains for finding invariants on values of variables

Some numerical domains

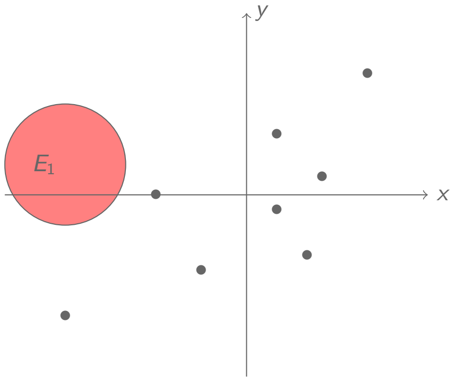
- Intervals (Cousot, Bourdoncle)
- Linear equalities (Karr)
- Polyhedra i.e. linear inequalities (Cousot, Halbwachs)
- Congruences and linear congruences (Granger)
- Octagons (Mine)
- Linear templates (Sankaranarayanan, Sipma, Manna)
- Zonotopes (Goubault, Putot)
- Non-linear templates (Adjé, Gaubert, Goubault, Seidl, Gawlitza)

Numerical abstract domains: example

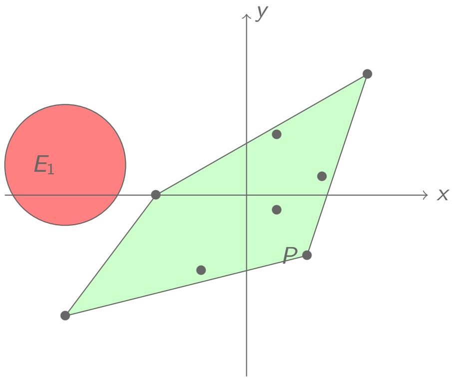


- Concrete set of values (x_i, y_i): program executions

Numerical abstract domains: example

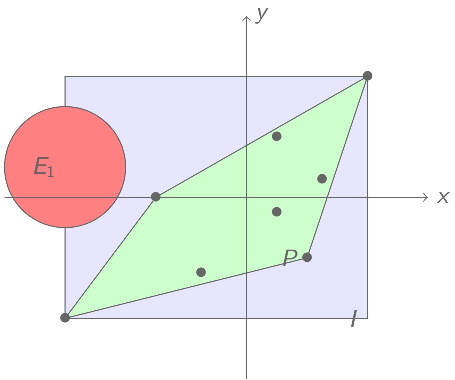


- Concrete set of values (x_i, y_i): program executions
- Forbidden zone E_1: is the program safe with respect to E_1?

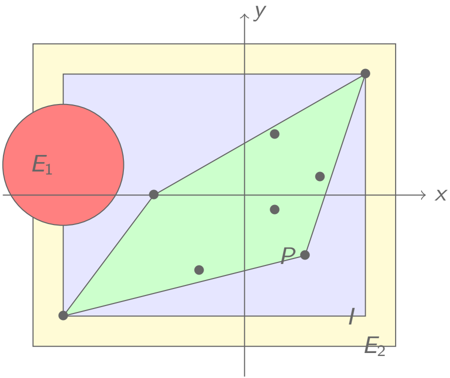


- Concrete set of values (x_i, y_i): program executions
- Forbidden zone E_1: is the program safe with respect to E_1?
- Polyhedra abstraction: proves the program is safe wrt E_1: $P \cap E_1 = \emptyset$

Numerical abstract domains: example



- Concrete set of values (x_i, y_i): program executions
- Forbidden zone E_1: is the program safe with respect to E_1?
- Polyhedra abstraction: proves the program is safe wrt E_1: $P \cap E_1 = \emptyset$
- Interval abstraction: cannot prove the program is safe wrt E_1 (false alarm): $I \cap E_1 \neq \emptyset$



- Concrete set of values (x_i, y_i): program executions
- Forbidden zone E_1: is the program safe with respect to E_1?
- Polyhedra abstraction: proves the program is safe wrt E_1: $P \cap E_1 = \emptyset$
- Interval abstraction: cannot prove the program is safe wrt E_1 (false alarm): $I \cap E_1 \neq \emptyset$
- Both abstractions prove the program is unsafe wrt to E_2: $P \subseteq I \subseteq E_2$

We are interested in local numerical invariants = at each program point, properties/values of variables true for all executions, and if possible the strongest properties

Example

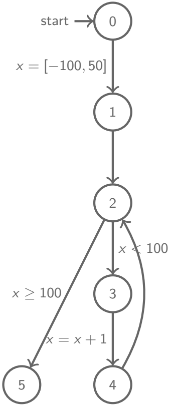
```
void main() {
  int x=[-100,50];
  // X = [-100,50]
  while (x<100) {
    // X = [-100,99]
    x=x+1;
    // X = [-99,100]
  }
   // X = [100,100]
}
```

Forward collecting semantics - informally

- We construct the control flow graph of a program, as a transition system $\mathcal{T} = (S, i, E, Tran)$

Example

```
void main()
{ // [0]
 int x=[-100,50];//[1]
 while /*[2]*/(x<100)
 {          // [3]
   x=x+1;   // [4]
 }          // [5]
}
```



Forward collecting semantics - informally

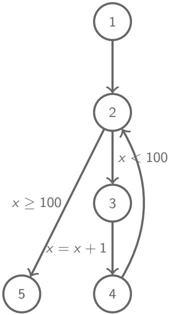
- We associate to each control point s_i of $\mathcal{T}$ an "equation" in associated variables X_i, which represent the sets of values that program variables can take, at control points s_i, and collect values coming from all paths:

$$X_i = \bigcup_{s_j \in S \mid (s_j, t, s_i) \in Tran} [\![t]\!] X_j$$

where $[\![t]\!]$ is the interpretation of the transition t (/transfer function), seen as a function from the set of values of variables to itself

Example

```
void main()
{ // [0]
  int x=[-100,50]; //[1]
  while /*[2]*/(x<100)
  {          // [3]
     x=x+1;  // [4]
  }          // [5]
}
```



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

Forward collecting semantics - informally

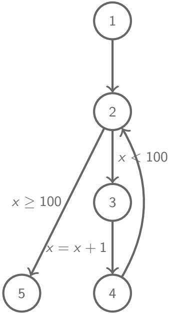
- We associate to each control point s_i of $\mathcal{T}$ an "equation" in associated variables X_i, which represent the sets of values that program variables can take, at control points s_i, and collect values coming from all paths:

$$X_i = \bigcup_{s_j \in S | (s_j, t, s_i) \in Tran} [\![t]\!] X_j$$

- The solutions of this system $X = F(X)$ are the local invariants at control points s_i ("property" always true for all possible executions)
  - interested in the strongest properties = the smallest fixpoint

Example

```
void main()
{ // [0]
 int x=[-100,50];//[1]
 while /*[2]*/(x<100)
 {          // [3]
   x=x+1;   // [4]
 }          // [5]
}
```



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

Concrete transfer functions for arithmetic expressions

- At each control point $s \in S$, we compute a set of environments $\Sigma \in \wp(Loc \rightarrow \mathbf{Z})$ reachable for some execution path
- We define the set of values an arithmetic expression can take:
  - $[\![n]\!]\Sigma = \{n\}$
  - $[\![X]\!]\Sigma = \Sigma(X)$ where we write $\Sigma(X) = \{\sigma(X) \mid \sigma \in \Sigma\}$
  - $[\![a_0 + a_1]\!]\Sigma = \{x + y \mid x = [\![a_0]\!]\sigma,\ y = [\![a_1]\!]\sigma,\ \sigma \in \Sigma\}$
  - $[\![a_0 - a_1]\!]\Sigma = \{x - y \mid x = [\![a_0]\!]\sigma,\ y = [\![a_1]\!]\sigma,\ \sigma \in \Sigma\}$
  - $[\![a_0 * a_1]\!]\Sigma = \{x * y \mid x = [\![a_0]\!]\sigma,\ y = [\![a_1]\!]\sigma, \sigma \in \Sigma\}$
- Then: $[\![X := a]\!]\Sigma = \Sigma[[\![a]\!]\Sigma/X]$ where $\Sigma[[\![a]\!]\Sigma/X] = \{\sigma[[\![a]\!]\sigma/X] \mid \sigma \in \Sigma\}$
- Conditionals
  - $[\![true]\!]\Sigma = \Sigma$
  - $[\![false]\!]\Sigma = \bot$ (where $\bot$ represents value "bottom" for the environments), i.e. $\bot(x) = \bot$ for all x
  - $[\![a_0 = a_1]\!]\Sigma = \Sigma_2$ with $\Sigma_2 \subseteq \Sigma$ is the set of environments $\sigma \in \Sigma$ such that $[\![a_0 = a_1]\!]\sigma = true$
  - Similarly for $a_0 < a_1$ etc.

Give a meaning to the smallest solution of the fixpoint equations $X = F(X)$ given by

$$X_i = \bigcup_{s_j \in S | (s_j, t, s_i) \in Tran} [\![t]\!] X_j?$$

Give a meaning to the smallest solution of the fixpoint equations $X = F(X)$ given by

$$X_i = \bigcup_{s_j \in S \mid (s_j, t, s_i) \in Tran} [\![t]\!] X_j?$$

Tarski's theorem

This least fixpoint exists and is unique because

- $D = (\wp(Loc \to \mathbf{Z}), \subseteq, \cup, \cap, \emptyset, (Loc \to \mathbf{Z}))$ is a complete lattice
- F is monotonic in this lattice $(\forall d, d' \in D,\ d \subseteq d' \Rightarrow f(d) \subseteq f(d'))$

Give a meaning to the smallest solution of the fixpoint equations $X = F(X)$ given by

$$X_i = \bigcup_{s_j \in S \mid (s_j, t, s_i) \in Tran} [\![t]\!] X_j ?$$

Tarski's theorem

This least fixpoint exists and is unique because

- $D = (\wp(Loc \to \mathbf{Z}), \subseteq, \cup, \cap, \emptyset, (Loc \to \mathbf{Z}))$ is a complete lattice
- F is monotonic in this lattice ($\forall d, d' \in D,\ d \subseteq d' \Rightarrow f(d) \subseteq f(d')$)

Kleene's iteration

Resolution by increasing iterations

$$fix(F) = \bigsqcup_{n \in \mathbf{N}} F^n(\bot)$$

is the smallest fixed point of F, because F is continuous on a CPO (monotonic and $\forall d_0 \subseteq d_1 \subseteq \ldots \subseteq d_n \subseteq \ldots$ of D: $\bigcup_{n \in \mathbf{N}} F(d_n) = F\left(\bigcup_{n \in \mathbf{N}} d_n\right)$)

A partial order $(P, \leq)$ is composed of:

- a set P and a binary relation $\leq \subseteq P \times P$ such that
- $\leq$ is reflexive: $\forall p \in P, p \leq p$
- $\leq$ is transitive: $\forall p, q, r \in P, p \leq q \& q \leq r \implies p \leq r$
- $\leq$ is anti-symetric: $\forall p, q \in P, p \leq q \& q \leq r \implies p = q$

Ex.: $(\wp(S), \subseteq)$

Upper bounds, Lattice, Completeness

Upper/lower bounds

- p is an upper bound of $X \subseteq P$ if $\forall q \in X, q \leq p$
- p is a (the!) least upper bound (lub, sup etc.) if:
  - p is an upper bound of X
  - for all upper bounds q of X, $p \leq q$
- similarly, lower bound and greatest lower bound (glb, inf etc.)

Lattice

- A lattice is a partial order P admitting a lub and a glb for all $X \subseteq P$ containing two elements (hence for any non empty finite set X)
- A complete partial order (cpo) is a partial order P where all ω-chains $p_0 \leq p_1 \leq \ldots \leq p_n \leq \ldots$ of P admit a lub
- In general, we suppose that a cpo also has a minimal element, denoted $\perp$
- A lattice is a complete lattice if all subsets admit a lub (and hence a glb).

Ex.: in $(\wp(S), \subseteq)$, all X admit a lub (the classical set union) and a glb (the classical set intersection). It is a complete lattice ($\perp = \emptyset$, $\top = S$).

We want the least fixpoint of the system $X = F(X)$ given by

$$F\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} \{-100,\dots,50\} \\ x_1 \cup x_4 \\ ]-\infty,99] \cap x_2 \\ x_3+1 \\ [100,+\infty[\cap x_2 \end{pmatrix} = \begin{pmatrix} F_1(x_1,\dots,x_5) \\ F_2(x_1,\dots,x_5) \\ \dots \\ \dots \\ F_5(x_1,\dots,x_5) \end{pmatrix}$$

Chaotic iterations

- classical Kleene iteration $X^0 = \bot, X^1 = F(X^0), \dots, X^{k+1} = X^k \cup F(X^k)$ is very slow
- we can "locally" iterate on any of the F_is, as long as all F_j are evaluated, potentially, an infinite number of times
- we use here the (classical too) iteration

$$\begin{array}{rcl} x_1^{k+1} & = & F_1(x_1^k,\dots,x_4^k,x_5^k) \\ x_2^{k+1} & = & F_2(x_1^{k+1},x_2^k,\dots,x_5^k) \\ \dots & & \\ x_5^{k+1} & = & F_5(x_1^{k+1},\dots,x_4^{k+1},x_5^k) \end{array}$$

Solution of fixpoint computations

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

$$
\begin{aligned}
x_1^0 &= \bot \\
x_2^0 &= \bot \\
x_3^0 &= \bot \\
x_4^0 &= \bot \\
x_5^0 &= \bot
\end{aligned}
$$

Solution of fixpoint computations

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

$$\begin{array}{ll}
x_1^0 = \perp & x_1^1 = \{-100, \ldots, 50\} \\
x_2^0 = \perp & x_2^1 = \{-100, \ldots, 50\} \\
x_3^0 = \perp & x_3^1 =]-\infty, 99] \cap \{-100, \ldots, 50\} = \{-100, \ldots, 50\} \\
x_4^0 = \perp & x_4^1 = \{-100, \ldots, 50\} + 1 = \{-99, \ldots, 51\} \\
x_5^0 = \perp & x_5^1 = [100, +\infty[\cap \{-100, \ldots, 50\} = \perp
\end{array}$$

Solution of fixpoint computations

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

$$
\begin{aligned}
x_1^1 &= \{-100, \ldots, 50\} \\
x_2^1 &= \{-100, \ldots, 50\} \\
x_3^1 &= \{-100, \ldots, 50\} \\
x_4^1 &= \{-99, \ldots, 51\} \\
x_5^1 &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^2 &= \{-100, \ldots, 50\} \\
x_2^2 &= \{-100, \ldots, 50\} \cup \{-99, \ldots, 51\} = \{-100, \ldots, 51\} \\
x_3^2 &=]-\infty, 99] \cap \{-100, \ldots, 51\} = \{-100, \ldots, 51\} \\
x_4^2 &= \{-100, \ldots, 51\} + 1 = \{-99, \ldots, 52\} \\
x_5^2 &= [100, +\infty[\cap \{-100, \ldots, 51\} = \bot
\end{aligned}
$$

Solution of fixpoint computations

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

$$\begin{aligned}
x_1^1 &= \{-100, \ldots, 50\} & x_1^2 &= \{-100, \ldots, 50\}\\
x_2^1 &= \{-100, \ldots, 50\} & x_2^2 &= \{-100, \ldots, 51\}\\
x_3^1 &= \{-100, \ldots, 50\} & x_3^2 &= \{-100, \ldots, 51\}\\
x_4^1 &= \{-99, \ldots, 51\} & x_4^2 &= \{-99, \ldots, 52\}\\
x_5^1 &= \perp & x_5^2 &= \perp
\end{aligned}$$

$k < 50$

$$\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\}\\
x_2^{k+1} &= \{-100, \ldots, 50\} \cup \{-99, \ldots, 50+k\} = \{-100, \ldots, 50+k\}\\
x_3^{k+1} &=]-\infty, 99] \cap \{-100, \ldots, 50+k\} = \{-100, \ldots, 50+k\}\\
x_4^{k+1} &= \{-100, \ldots, 50+k\} + 1 = \{-99, \ldots, 51+k\}\\
x_5^{k+1} &= [100, +\infty[\cap \{-100, \ldots, 50+k\} = \perp
\end{aligned}$$

Solution of fixpoint computations

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

$$
\begin{aligned}
x_1^1 &= \{-100, \ldots, 50\} \\
x_2^1 &= \{-100, \ldots, 50\} \\
x_3^1 &= \{-100, \ldots, 50\} \\
x_4^1 &= \{-99, \ldots, 51\} \\
x_5^1 &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^2 &= \{-100, \ldots, 50\} \\
x_2^2 &= \{-100, \ldots, 51\} \\
x_3^2 &= \{-100, \ldots, 51\} \\
x_4^2 &= \{-99, \ldots, 52\} \\
x_5^2 &= \bot
\end{aligned}
$$

$k < 50$

$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= \{-100, \ldots, 50 + k\} \\
x_3^{k+1} &= \{-100, \ldots, 50 + k\} \\
x_4^{k+1} &= \{-99, \ldots, 51 + k\} \\
x_5^{k+1} &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^{51} &= \{-100, \ldots, 50\} \\
x_2^{51} &= \{-100, \ldots, 50\} \cup \{-99, \ldots, 100\} = \{-100, \ldots, 100\} \\
x_3^{51} &=]-\infty, 99] \cap \{-100, \ldots, 100\} = \{-100, \ldots, 99\} \\
x_4^{51} &= \{-100, \ldots, 51\} + 1 = \{-99, \ldots, 100\} \\
x_5^{51} &= [100, +\infty[\cap \{-100, \ldots, 100\} = 100
\end{aligned}
$$

Solution of fixpoint computations

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

$$\begin{aligned}
x_1^1 &= \{-100, \dots, 50\} \\
x_2^1 &= \{-100, \dots, 50\} \\
x_3^1 &= \{-100, \dots, 50\} \\
x_4^1 &= \{-99, \dots, 51\} \\
x_5^1 &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^2 &= \{-100, \dots, 50\} \\
x_2^2 &= \{-100, \dots, 51\} \\
x_3^2 &= \{-100, \dots, 51\} \\
x_4^2 &= \{-99, \dots, 52\} \\
x_5^2 &= \bot
\end{aligned}$$

$k < 50$

$$\begin{aligned}
x_1^{k+1} &= \{-100, \dots, 50\} \\
x_2^{k+1} &= \{-100, \dots, 50 + k\} \\
x_3^{k+1} &= \{-100, \dots, 50 + k\} \\
x_4^{k+1} &= \{-99, \dots, 51 + k\} \\
x_5^{k+1} &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^{51} &= \{-100, \dots, 50\} \\
x_2^{51} &= \{-100, \dots, 100\} \\
x_3^{51} &= \{-100, \dots, 99\} \\
x_4^{51} &= \{-99, \dots, 100\} \\
x_5^{51} &= 100
\end{aligned}
\qquad
k \geq 51, X^k = X^{51}$$

Partial conclusion

- We were lucky, as there is no reason that we can solve these semantic equations in general in finite time
- $\longrightarrow$ abstract the semantics!

Abstract Interpretation (Cousot & Cousot 77)

- A theory of semantics approximation
- The simple/elegant framework relies on Galois connections between complete lattices (abstraction by intervals for instance)
- But weaker structures also used

Galois connections

- Let $\mathcal{C}$ be a complete lattice of concrete properties: (e.g. $(\wp(\mathbf{Z}), \subseteq)$)
- Let $\mathcal{A}$ be a complete lattice of abstract properties: (e.g. $(S, \sqsubseteq)$)
- $\alpha : \mathcal{C} \to \mathcal{A}$ (abstraction) and $\gamma : \mathcal{A} \to \mathcal{C}$ (concretisation) two monotonic functions (we could forget this hypothesis) such that

$$\alpha(x) \leq_{\mathcal{A}} y \;\Leftrightarrow\; x \leq_{\mathcal{C}} \gamma(y)$$

Example: lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\perp$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$
- Sup : $[a, b] \cup [c, d] = [\min(a, c), \max(b, d)]$
- Inf : $[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$
- Complete

$$\bigcup_{i \in I} [a_i, b_i] = [inf_{i \in I}\ a_i, sup_{i \in I}\ b_i]$$

Abstraction

$$\begin{array}{rcccl} \alpha & : & \wp(\mathbb{R}) & \rightarrow & \mathcal{I} \\ & & S & \rightarrow & [\mathit{inf}\ S, \mathit{sup}\ S] \end{array}$$

(the *inf* and *sup* are taken in $\mathbb{R} \cup \{-\infty, \infty\}$)

Concretisation

$$\begin{array}{rcccl} \alpha & : & \mathcal{I} & \rightarrow & \wp(\mathbb{R}) \\ & & [a, b] & \rightarrow & \{x \in \mathbb{R} \mid a \leq x \leq b\} \end{array}$$

(a and b are potentially infinite)

Properties of Galois connections

- We can check that we have the following properties:
  (1) $\alpha \circ \gamma(x) \leq_{\mathcal{A}} x$
  (2) $y \leq_{\mathcal{C}} \gamma \circ \alpha(y)$
  (1) and (2) and α and γ monotone are equivalent conditions to the fact that (α, γ) is a Galois connection (exercise!).
- α and γ are said to be quasi-inverses: as for inverses, one can be uniquely defined in term of the other ($\bigcap$ is glb, $\bigcup$ is lub)

$$\begin{array}{rcl} \alpha(x) & = & \bigcap\{y \mid x \leq_{\mathcal{C}} \gamma(y)\} \\ \gamma(x) & = & \bigcup\{y \mid \alpha(y) \leq_{\mathcal{A}} x\} \end{array}$$

- In other terms:
  - α give the most precise abstract value representing a given concrete property
  - γ gives the semantics of abstract values, in terms of concrete properties

Best abstraction of a transfer function F

For all concrete functionals $F : \mathcal{C} \rightarrow \mathcal{C}$ (for example, the collecting semantics), we define an abstract functional $F^\sharp : \mathcal{A} \rightarrow \mathcal{A}$ by

$$F^\sharp(y) = \alpha \circ F \circ \gamma(y)$$

It is the best possible abstraction of F.

Sound abstraction

In practice, α and/or γ are not computable (algorithmically speaking). We use in general an over-approximation F' such that $F^\sharp(y) \leq_{\mathcal{A}} F'(y)$.

Abstraction of the collecting semantics

- The abstract values are abstract environments $\sigma^\sharp : Loc \rightarrow S$
- Non relational abstraction: we abstract $\wp(Loc \rightarrow \mathbf{Z})$ by $Loc \rightarrow \wp(\mathbf{Z})$
  - We forget the potential relations between the values of variables
  - As Loc is finite (say, of cardinal $n \in \mathbf{N}$), $Loc \rightarrow S \equiv S^n$
  - S is a complete lattice $\Rightarrow$ S^n is a complete lattice defined componentwise

Determining the best interval addition

- Let $+ : \wp(\mathbb{R}) \times \wp(\mathbb{R}) \to \wp(\mathbb{R})$ be the standard, real number addition, lifted onto sets of real numbers
- We want to find its best abstraction on intervals

$$\oplus : \mathcal{I} \to \mathcal{I}$$

Determining the best interval addition

- Let $+ : \wp(\mathbb{R}) \times \wp(\mathbb{R}) \to \wp(\mathbb{R})$ be the standard, real number addition, lifted onto sets of real numbers
- We want to find its best abstraction on intervals

$$\oplus : \mathcal{I} \to \mathcal{I}$$

We compute:

$$\begin{array}{rcl}
[a, b] \oplus [c, d] & = & \alpha\left(\gamma\left([a, b]\right) + \gamma\left([c, d]\right)\right) \\
& = & \alpha\left(\{x \mid a \leq x \leq b\} + \{y \mid c \leq y \leq d\}\right) \\
& = & \alpha\left(\{x + y \mid a \leq x \leq b,\ c \leq y \leq d\}\right) \\
& = & \alpha\left(\{x + y \mid a + c \leq x \leq b + d\}\right) \\
& = & [a + c, b + d]
\end{array}$$

($+$ is naturally extended to infinite numbers - we never have to write $\infty + (-\infty)$ in the above)

Determining the best interval addition

- Let $+ : \wp(\mathbb{R}) \times \wp(\mathbb{R}) \to \wp(\mathbb{R})$ be the standard, real number addition, lifted onto sets of real numbers
- We want to find its best abstraction on intervals

$$\oplus : \mathcal{I} \to \mathcal{I}$$

We compute:

$$\begin{array}{rcl}
[a,b] \oplus [c,d] & = & \alpha\left(\gamma\left([a,b]\right) + \gamma\left([c,d]\right)\right) \\
& = & \alpha\left(\{x \mid a \leq x \leq b\} + \{y \mid c \leq y \leq d\}\right) \\
& = & \alpha\left(\{x + y \mid a \leq x \leq b,\ c \leq y \leq d\}\right) \\
& = & \alpha\left(\{x + y \mid a + c \leq x \leq b + d\}\right) \\
& = & [a + c, b + d]
\end{array}$$

($+$ is naturally extended to infinite numbers - we never have to write $\infty + (-\infty)$ in the above)

Exercise: what if one of the arguments is $\perp$, $\top$? What about multiplication/division?

Transfer functions for conditionals and loops

- $[\![true]\!]\sigma^\sharp = \sigma^\sharp$
- $[\![false]\!]\sigma^\sharp = \bot$
- $[\![x = y]\!]\sigma^\sharp(z) = \begin{cases} \sigma^\sharp(x) \cap \sigma^\sharp(y) & \text{if } z = x, y \\ \sigma^\sharp(z) & \text{if } z \neq x, y \end{cases}$
- $[\![x <= y]\!]\sigma^\sharp(z) = \begin{cases} \sigma^\sharp(x) \cap]-\infty, sup\ \sigma^\sharp(y)] & \text{if } z = x \\ \sigma^\sharp(y) \cap [inf\ \sigma^\sharp(x), \infty[& \text{if } z = y \\ \sigma^\sharp(z) & \text{if } z \neq x, y \end{cases}$
- etc.

How do we interpret $x + y = 5$??

Example: interpret $x + y = 5$ starting with $x = [0, 10]$ and $y = [3, 6]$:

- Look at $5 - y$: should be x. But $5 - y = [-1, 2]$.
  This improves bounds on x: $x \leftarrow x \cap (5 - y) = [0, 2]$
- Look at $5 - x$: should be y. But $5 - x = [-5, 5]$.
  This improves bounds on y: $y \leftarrow y \cap (5 - x) = [3, 5]$

Example: interpret $x + y = 5$ starting with $x = [0, 10]$ and $y = [3, 6]$:

- Look at $5 - y$: should be x. But $5 - y = [-1, 2]$.
  This improves bounds on x: $x \leftarrow x \cap (5 - y) = [0, 2]$
- Look at $5 - x$: should be y. But $5 - x = [-5, 5]$.
  This improves bounds on y: $y \leftarrow y \cap (5 - x) = [3, 5]$

We carry on: can we improve bounds on x and bounds on y?:

- Look at $5 - y$: should be x. But $5 - y = [0, 2]$.
  We try to improve bounds on x: $x \leftarrow x \cap (5 - y) = [0, 2]$ (stable!)
- Look at $5 - x$: should be y. But $5 - x = [3, 5]$.
  We try to improve bounds on y: $y \leftarrow y \cap (5 - x) = [3, 5]$ (stable!)

Constraint solving is the computation of a gfp under the initial values of the variables: local decreasing iterations to solve conditionals accurately!

When we have a Galois connection (α, γ) between a concrete domain $\mathcal{C}$ and an abstract domain $\mathcal{A}$, we have, for all concrete functionals $F : \mathcal{C} \rightarrow \mathcal{C}$:

$$\alpha(\mathit{lfp}(F)) \leq_{\mathcal{A}} \mathit{lfp}(F^{\sharp})$$

or, equivalently:

$$\mathit{lfp}(F) \leq_{\mathcal{C}} \gamma \circ \mathit{lfp}(F^{\sharp})$$

Hence, the abstract computation gives an over-approximation of the concrete invariants.

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

$k < 50$

$$
\begin{aligned}
x_1^{k+1} &= [-100, 50] \\
x_2^{k+1} &= [-100, 50 + k] \\
x_3^{k+1} &= [-100, 50 + k] \\
x_4^{k+1} &= [-99, 51 + k] \\
x_5^{k+1} &== \perp
\end{aligned}
$$

$k \geq 51, X^k = X^{51}$

$$
\begin{aligned}
x_1^{51} &= [-100, 50] \\
x_2^{51} &= [-100, 100] \\
x_3^{51} &= [-100, 99] \\
x_4^{51} &= [-99, 100] \\
x_5^{51} &= 100
\end{aligned}
$$

Same Kleene iterations in intervals ... can be quite long and even untractable!

Acceleration and terminaison by widening

- Step $k+1$, $k<10$ for instance, followed by one or more widening steps

$$X^{k+1} = X^k \nabla F(X^k)$$

For some $m > 10$, $\longrightarrow X^m$ post-fixed point of F (such that $F(X^m) \subseteq X^m$).

- Followed by a finite number of narrowing steps

$$X^{k+1} = X^k \Delta F(X^k)$$

for $k > m$,

- Converges in finite time to a fixed point of F but not necessarily the smallest one.

In general, one widening step per cycle in the control flow graph (entry of a loop for instance)

Widening:

$$[a,b]\nabla[c,d] = [e,f] \text{ with } \quad e = \begin{cases} a & \text{if } a \leq c \\ -\infty & \text{otherwise} \end{cases} \quad \text{and}$$
$$f = \begin{cases} b & \text{if } d \leq b \\ \infty & \text{otherwise,} \end{cases}$$

Narrowing:

$$[a,b]\Delta[c,d] = [e,f] \text{ with } \quad e = \begin{cases} c & \text{if } a = -\infty \\ a & \text{otherwise} \end{cases} \quad \text{and}$$
$$f = \begin{cases} d & \text{if } b = \infty \\ b & \text{otherwise} \end{cases}$$

Example with widening/narrowing

Widening phase: $k \geq 10$

$$
\begin{aligned}
x_1^{k+1} &= [-100, 50] \\
x_2^{k+1} &= x_2^k \nabla (x_1^{k+1} \cup x_4^k) \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}
$$

$$
\begin{aligned}
x_2^{11} &= [-100, 60] \nabla [-100, 61] = [-100, \infty[\\
x_3^{11} &=]-\infty, 99] \cap [-100, \infty[= [-100, 99] \\
x_4^{11} &= [-99, 100] \\
x_5^{11} &= [100, +\infty[\cap [-100, \infty[= [100, +\infty[
\end{aligned}
$$

Example with widening/narrowing

Widening phase: $k \geq 10$

$$
\begin{aligned}
x_1^{k+1} &= [-100, 50] \\
x_2^{k+1} &= x_2^k \nabla (x_1^{k+1} \cup x_4^k) \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}
$$

$$
\begin{aligned}
x_2^{11} &= [-100, \infty[& \quad x_2^{12} &= [-100, \infty[\nabla([-100, 50] \cup [-99, 100]) = [-100, \infty[\\
x_3^{11} &= [-100, 99] & \quad x_3^{12} &= [-100, 99] \\
x_4^{11} &= [-99, 100] & \quad x_4^{12} &= [-99, 100] \\
x_5^{11} &= [100, +\infty[& \quad x_5^{12} &= [100, +\infty[
\end{aligned}
$$

$X^{11} = X^{12}$ is a postfix point

Example with widening/narrowing

Widening phase: $k \geq 10$

$$\begin{aligned}
x_1^{k+1} &= [-100, 50] \\
x_2^{k+1} &= x_2^k \nabla (x_1^{k+1} \cup x_4^k) \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}$$

Narrowing phase: $k \geq 12$

$$\begin{aligned}
x_1^{k+1} &= [-100, 50] \\
x_2^{k+1} &= x_2^k \Delta (x_1^{k+1} \cup x_4^k) \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}$$

$$\begin{aligned}
x_2^{11} &= [-100, \infty[\\
x_3^{11} &= [-100, 99] \\
x_4^{11} &= [-99, 100] \\
x_5^{11} &= [100, +\infty[
\end{aligned}
\qquad
\begin{aligned}
x_2^{12} &= [-100, \infty[\\
x_3^{12} &= [-100, 99] \\
x_4^{12} &= [-99, 100] \\
x_5^{12} &= [100, +\infty[
\end{aligned}$$

$X^{11} = X^{12}$ is a postfix point

$$\begin{aligned}
x_2^{13} &= [-100, \infty[\Delta ([-100, 50] \cup [-99, 100]) = [-100, 100] \\
x_3^{13} &=]-\infty, 99] \cap [-100, 100] = [-100, 99] \\
x_4^{13} &= [-99, 100] \\
x_5^{13} &= [100, +\infty[\cap [-100, 100] = [100, 100]
\end{aligned}$$

Example with widening/narrowing

Widening phase: $k \geq 10$

$$\begin{aligned}
x_1^{k+1} &= [-100, 50] \\
x_2^{k+1} &= x_2^k \nabla (x_1^{k+1} \cup x_4^k) \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}$$

Narrowing phase: $k \geq 12$

$$\begin{aligned}
x_1^{k+1} &= [-100, 50] \\
x_2^{k+1} &= x_2^k \Delta (x_1^{k+1} \cup x_4^k) \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}$$

$$\begin{aligned}
x_2^{11} &= [-100, \infty[\\
x_3^{11} &= [-100, 99] \\
x_4^{11} &= [-99, 100] \\
x_5^{11} &= [100, +\infty[
\end{aligned}
\qquad
\begin{aligned}
x_2^{12} &= [-100, \infty[\\
x_3^{12} &= [-100, 99] \\
x_4^{12} &= [-99, 100] \\
x_5^{12} &= [100, +\infty[
\end{aligned}$$

$X^{11} = X^{12}$ is a postfix point

$$\begin{aligned}
x_2^{13} &= [-100, 100] \\
x_3^{13} &= [-100, 99] \\
x_4^{13} &= [-99, 100] \\
x_5^{13} &= [100, 100]
\end{aligned}
\qquad
\begin{aligned}
x_2^{14} &= [-100, 100] \\
x_3^{14} &= [-100, 99] \\
x_4^{14} &= [-99, 100] \\
x_5^{14} &= [100, 100]
\end{aligned}$$

$X^{13} = X^{14}$ is a fixpoint (the least fixpoint here)

In a general abstract domain $\mathcal{A}$:

Widening

- For all $x, y \in \mathcal{A}$, $x, y \leq_A x \nabla y$
- There is no infinite strictly increasing sequence:
  $x_0 < x_1 = x_0 \nabla y_0 < x_2 = x_1 \nabla y_1 < \ldots < x_{n+1} = x_n \nabla y_n < \ldots$
  whenever $y_0 \leq_A y_1 \leq_A \ldots \leq_A y_n \leq_A \ldots$
- This ensures that replacing $\cup$ by ∇ in Kleene iteration sequence makes it convergent to a post-fixed point of a functional.

Narrowing

- for $x, y \in \mathcal{A}$ with $y \leq_A x$, $y \leq_A x \Delta y \leq_A x$
- There is no infinite decreasing sequence:
  $\ldots < x_{n+1} = x_n \Delta y_n < x_2 = x_1 \Delta y_1 < x_1 = x_0 \Delta y_0 < x_0$
  whenever $\ldots \leq_A y_n \leq_A \ldots \leq_A y_1 \leq_A y_0$
- This implies that, from a post-fixed point, the Kleene iteration sequence where we replace $\cup$ by Δ, converges to a fixed point of F.

Partial conclusions

- Widening/narrowing can be extremely fast
- But not so easy to tune and to design
- For instance, you might want at least to "unroll" loops a certain number of times (without taking any union) before actually performing Kleene iteration sequence with widenings and narrowings

[demo Fluctuat `ex_loop.c`]

Relational domains

Non-relational domains

- such as intervals, signs etc. do not represent relations between variables
- as a consequence, the abstraction can be very coarse, e.g., in intervals, if $x = [a, b]$, $x - x$ is interpreted as $[a, b] - [a, b] = [a - b, b - a]$ and not 0!

Need for relational domains

```
X = [0,10];  //2
Y = [0,10];  //3
S = X-Y;  //4
if (S>=2)  //5
  Y = Y+2;
// 6
```

[demo interproc `ex_zones.txt`]

In intervals:
Y in [0,12] at 6

Using relations:
At 5: X-Y>=2 hence Y<=8
So at 6, Y in [0,10]

Relational domains?

A simple solution (Mine 01):

- Zones: represent $v_i - v_j \leq c_{ij}$ (with v_0 dummy variable, always equal to zero, so as to represent bounds of variables as well)
  A new numerical abstract domain based on difference-bound matrices. A. Miné. In PADO, volume 2053 of LNCS, May 2001.
- Octagons: zones plus relations of the form $v_i + v_j \leq c_{ij}$ The octagon abstract domain. A. Miné. In AST'01, Oct. 2001

Good enough?

- Computationally OK, sometimes expansive (some operations in $O(n^3)$)
- With "packing of variables", this is one ingredient of the success of Astrée for instance
- Enough for part of our `fft` proof, but not quite; not enough for linear filters etc. $\rightarrow$ general linear inequalities

General polyhedra

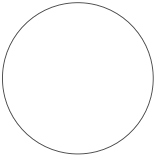
Automatic discovery of linear restraints among variables of a program. P. Cousot and N. Halbwachs. POPL 78.

Domain $\mathcal{P}$:

- Abstract values are systems of linear inequalities on values of variables, i.e. of the form:

$$\left\{\begin{array}{lcl} a_{11}v_1 + a_{12}v_2 + \ldots + a_{1n}v_n & \leq & c_1 \\ a_{21}v_1 + a_{22}v_2 + \ldots + a_{2n}v_n & \leq & c_2 \\ \ldots & & \\ a_{k1}v_1 + a_{k2}v_2 + \ldots + a_{kn}v_n & \leq & c_k \end{array}\right.$$

- Concretisation operator $\gamma : \mathcal{P} \to \wp(\mathbb{R}^n)$ is obvious
- Abstraction operator?



General polyhedra

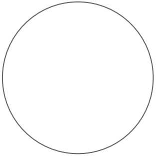
Automatic discovery of linear restraints among variables of a program. P. Cousot and N. Halbwachs. POPL 78.

Domain $\mathcal{P}$:

- Abstract values are systems of linear inequalities on values of variables, i.e. of the form:

$$\left\{\begin{array}{lcl} a_{11}v_1 + a_{12}v_2 + \ldots + a_{1n}v_n & \leq & c_1 \\ a_{21}v_1 + a_{22}v_2 + \ldots + a_{2n}v_n & \leq & c_2 \\ \ldots & & \\ a_{k1}v_1 + a_{k2}v_2 + \ldots + a_{kn}v_n & \leq & c_k \end{array}\right.$$

- Concretisation operator $\gamma : \mathcal{P} \to \wp(\mathbb{R}^n)$ is obvious
- Abstraction operator? None! (not a Galois connection)



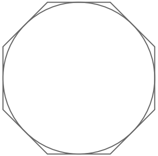
Automatic discovery of linear restraints among variables of a program. P. Cousot and N. Halbwachs. POPL 78.

Domain $\mathcal{P}$:

- Abstract values are systems of linear inequalities on values of variables, i.e. of the form:

$$\begin{cases} a_{11}v_1 + a_{12}v_2 + \ldots + a_{1n}v_n & \leq & c_1 \\ a_{21}v_1 + a_{22}v_2 + \ldots + a_{2n}v_n & \leq & c_2 \\ \ldots & & \\ a_{k1}v_1 + a_{k2}v_2 + \ldots + a_{kn}v_n & \leq & c_k \end{cases}$$

- Concretisation operator $\gamma : \mathcal{P} \to \wp(\mathbb{R}^n)$ is obvious
- Abstraction operator? None! (not a Galois connection)
  - The polyhedra lattice is not complete: strictly increasing chains with limit not a polyhedron (disc)
  - Best abstraction of the disc?



Partial conclusion

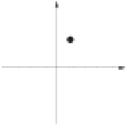
Polyhedra

- Fairly precise domain
- Most implementations use the double description method:
  - as a set of constraints (faces) : intersections are easy to compute
  - as the convex hull of a set of points+rays (generators): unions are easy to compute
- Potential exponential cost between the two representation...hence this domain is used for small programs, on 10 to 50 variables max

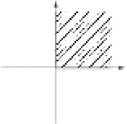
Possible answers to that:

- Linear templates: choose a finite set of normals to faces, use linear programming to compute the abstract functions
  S. Sankaranarayanan, H. Sipma, Z. Manna, Scalable Analysis of Linear Systems using Mathematical Programming. In VMCAI 2005, Volume 3385 of LNCS.
- In the following, the domain of zonotopes: no choice of faces needed, but a symmetry condition is added; simple and tractable representation. Similarly to polyhedra, we will not have best abstractions, but concretization based abstract interpretation works fine.

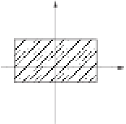
A partial zoo of numerical domains



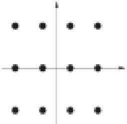
Constant Propagation
$X_i = c_i$
[Kil73]



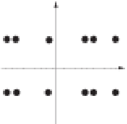
Signs
$X_i \geq 0,\ X_i \leq 0$
[CC76]



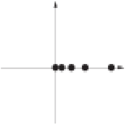
Intervals
$X_i \in [a_i, b_i]$
[CC76]



Simple Congruences
$X_i \equiv a_i\,[b_i]$
[Gra89, Gra97]

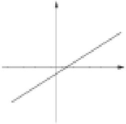


Interval Congruences
$X_i \in \alpha_i[a_i, b_i]$
[Mas93]

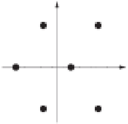


Power Analysis
$X_i \in \alpha_i^{a_i\mathbb{Z}+b_i}, \alpha_i^{[a_i,b_i]}$, etc.
[Mas01]

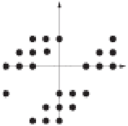
A partial zoo of numerical domains



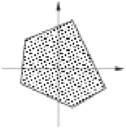
Linear Equalities
$\sum_i \alpha_{ij} X_i = \beta_j$
[Kar76]



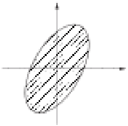
Linear Congruences
$\sum_i \alpha_{ij} X_i \equiv \beta_j \, [\gamma_j]$
[Gra91]



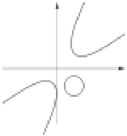
Trapezoidal Congruences
$X_i = \sum_j \lambda_j \alpha_{ij} + \beta_j$
[Mas92]



Polyhedra
$\sum_i \alpha_{ij} X_i \leq \beta_j$
[CH78]



Ellipsoids
$\alpha X^2 + \beta Y^2 + \gamma XY \leq \delta$
[Fer04b]



Varieties
$P_i(\vec{X}) = 0, \; P_i \in \mathbb{R}[\mathcal{V}]$
[RCK04a]

Some “commercial” abstract interpreters

- Polyspace (now at Mathworks)
  http://www.mathworks.fr/products/polyspace/
- WCET, Stack Analyzer, Astrée (ABSINT) http://www.absint.com
- Clousot: code contracts checking with abstract interpretation (Microsoft)
  http://research.microsoft.com/apps/pubs/?id=138696
- C global surveyor (NASA) http://ti.arc.nasa.gov/tech/rse/vandv/cgs/
  (used on flight software of Mars Path-Finder, Deep Space One, and Mars Exploration Rover)
  CodeHawk http://www.kestreltechnology.com/codehawk/codehawk.php
- Frama-C’s value analysis http://frama-c.com/value.html
- Fluctuat (proprietary tool but academic version upon request)
  http://www.lix.polytechnique.fr/Labo/Sylvie.Putot/fluctuat.html

Notes, links and exercising

We will focus in this lecture on numerical properties and abstract domains:

- Michael I. Schwartsbach's Lecture Notes on Static Analysis provide a larger view on non purely numerical analyses

Numerical abstract domains:

- The second chapter of Antoine Mine's Ph.D thesis provides an accessible introduction to abstract interpretation and classical numerical abstract domains
- Bertrand Jeannet's Interproc Analyzer and web interface allows one to try some classical abstract domains of the Apron library of abstract domains (in particular, intervals, octagons and polyhedra) on small examples
- Saarland University/Absint's PAG WWW allows one to follow the different steps of simple predefined analyses on small examples; then you can provide your own analysis as an abstract domain with its transfer functions. As a first exercice, you can for instance improve the predefined constant propagation analysis as proposed in this TD (also see slides 14 to 24 here for some notes on this predefined constant propagation analysis)

II. From Affine Arithmetic to zonotopic abstract domains

Affine forms

- Affine form for variable x:

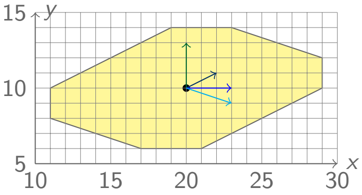
$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n, \; x_i \in \mathbb{R}$$

  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 $\hat{x}$:

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

Geometric concretization as zonotopes in $\mathbb{R}^p$

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



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 For instance multiplication:

$$\begin{aligned}\hat{x} \times \hat{y} \quad = \quad & (\alpha_0^x \alpha_0^y + \frac{1}{2}\sum_{i=1}^{n} |\alpha_i^x \alpha_i^y|) + \sum_{i=1}^{n}(\alpha_i^x \alpha_0^y + \alpha_i^y \alpha_0^x)\varepsilon_i + \\ & (\frac{1}{2}\sum_{i=1}^{n} |\alpha_i^x \alpha_i^y| + \sum_{1 \le i < j \le n} |\alpha_i^x \alpha_j^y + \alpha_j^x \alpha_i^y|)\varepsilon_{n+1}.\end{aligned}$$

- 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 For instance multiplication:

$$\begin{aligned}\hat{x} \times \hat{y} \quad = \quad & (\alpha_0^x \alpha_0^y + \frac{1}{2}\sum_{i=1}^{n} |\alpha_i^x \alpha_i^y|) + \sum_{i=1}^{n} (\alpha_i^x \alpha_0^y + \alpha_i^y \alpha_0^x)\varepsilon_i + \\ & (\frac{1}{2}\sum_{i=1}^{n} |\alpha_i^x \alpha_i^y| + \sum_{1 \le i < j \le n} |\alpha_i^x \alpha_j^y + \alpha_j^x \alpha_i^y|)\varepsilon_{n+1}.\end{aligned}$$

Example: $\hat{x} = \varepsilon_1 + \varepsilon_2$ and $\hat{y} = \varepsilon_2$, $\hat{x} \times \hat{y} = \frac{1}{2} + \frac{3}{2}\varepsilon_3 \in [-1, 2]$. However, the exact range here is [-0.25,2]; could use a better SDP formula...

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 For instance multiplication:

$$\begin{aligned} \hat{x} \times \hat{y} \quad = \quad & (\alpha_0^x \alpha_0^y + \frac{1}{2}\sum_{i=1}^{n} |\alpha_i^x \alpha_i^y|) + \sum_{i=1}^{n}(\alpha_i^x \alpha_0^y + \alpha_i^y \alpha_0^x)\varepsilon_i + \\ & (\frac{1}{2}\sum_{i=1}^{n} |\alpha_i^x \alpha_i^y| + \sum_{1 \le i < j \le n} |\alpha_i^x \alpha_j^y + \alpha_j^x \alpha_i^y|)\varepsilon_{n+1}. \end{aligned}$$

- Close to Taylor models of low degree (large ranges for static analysis)

Division

$$\frac{\hat{x}}{\hat{y}} = \left(\left(\frac{\alpha_0^x}{\alpha_0^y}(1+\mathrm{mid}(\mathbf{g}_y)) - 0.5 \sum_{i=1}^n \frac{|\alpha_i^x \alpha_i^y|}{(\alpha_0^y)^2}\right) + \sum_{i=1}^n \left(\frac{\alpha_i^x}{\alpha_0^y}(1+\mathrm{mid}(\mathbf{g}_y)) - \frac{\alpha_0^x \alpha_i^y}{(\alpha_0^y)^2}\right) \varepsilon_i + \left(0.5 \sum_{i=1}^n \frac{|\alpha_i^x \alpha_i^y|}{(\alpha_0^y)^2} + \sum_{1 \le i < j \le n} \frac{|\alpha_i^x \alpha_j^y + \alpha_j^x \alpha_i^y|}{(\alpha_0^y)^2} + \sum_{i=0}^n \frac{|\alpha_i^x|}{|\alpha_0^y|} \mathrm{dev}(\mathbf{g}_y) \right) \varepsilon_{n+1}$$

Square root

$$\sqrt{\hat{x}} = \sqrt{\alpha_0^x}(1+\mathrm{mid}(\mathbf{f}_x)) + \frac{1}{2} \sum_{i=1}^n \frac{\alpha_i^x}{\sqrt{\alpha_0^x}} \varepsilon_i + \sqrt{\alpha_0^x}\mathrm{dev}(\mathbf{f}_x)\, \varepsilon_{n+1},$$

where $\mathbf{f}_x$ is the enclosure of function f defined by $f(u) = \sqrt{1+u} - \left(1 + \frac{1}{2}u\right)$ when u takes the values greater than -1 that $\sum_{i=1}^n \frac{\alpha_i^x}{\alpha_0^x} \varepsilon_i$ can take.

Affine forms define implicit relations : example

Consider, with $a \in [-1, 1]$ and $b \in [-1, 1]$, the expressions

```
x = 1 + a + 2 * b;
y = 2 - a;
z = x + y - 2 * b;
```

- The representation as affine forms is $\hat{x} = 1 + \epsilon_1 + 2\epsilon_2$, $\hat{y} = 2 - \epsilon_1$, with noise symbols $\epsilon_1, \epsilon_2 \in [-1, 1]$
- This implies $\hat{x} \in [-2, 4]$, $\hat{y} \in [1, 3]$ (same as I.A.)
- It also contains implicit relations, such as $\hat{x} + \hat{y} = 3 + 2\epsilon_2 \in [1, 5]$ or $\boxed{\hat{z} = \hat{x} + \hat{y} - 2b = 3}$
- Whereas we get with intervals

$$\boxed{z = x + y - 2b \in [-3, 9]}$$

Implementation using floating-point numbers

- For implementation of affine forms, we do not have real but floating-point coefficients (possibly higher precision fp numbers using MPFR library)
- One solution is to compute each coefficient of the affine form with intervals of f.p. numbers with outward rounding
  - inaccurate because of intervals
- More accurate : keep point coefficients and handle uncertainty on these coefficients by creating new noise terms

Numerical abstract domain (in short!)

Concretization-based analysis

- Machine-representable abstract values X (affine sets)
- A concretization function γ_f defining the set of concrete values represented by an abstract value
- A partial order on these abstract values, induced by γ_f:
  $X \sqsubseteq Y \iff \gamma_f(X) \subseteq \gamma_f(Y)$

Abstract transfer functions

- Arithmetic operations: F is a sound abstraction of f iff

$$\forall x \in \gamma_f(X),\ f(x) \in \gamma_f(F(X))$$

- Set operations: join ($\cup$), meet ($\cap$), widening
  - no least upper bound / greatest lower bound on affine sets
  - (minimal) upper bounds / over-approximations of the intersection ...

and ... hopefully accurate and effective to compute!!!

Representation of zonotopes

- $\mathcal{M}(n, p)$: matrices with n lines and p columns of real coefficients
- A form expressing the set of values taken by p variables over n noise symbols ε_i, $1 \leq i \leq n$, can be represented by a matrix $A \in \mathcal{M}(n+1, p)$

Representation of zonotopes

- $\mathcal{M}(n,p)$: matrices with n lines and p columns of real coefficients
- A form expressing the set of values taken by p variables over n noise symbols ε_i, $1 \leq i \leq n$, can be represented by a matrix $A \in \mathcal{M}(n+1,p)$

Example

$$\begin{aligned} \hat{x} &= 20 - 4\varepsilon_1 + 2\varepsilon_3 + 3\varepsilon_4 && (1) \\ \hat{y} &= 10 - 2\varepsilon_1 + \varepsilon_2 - \varepsilon_4, && (2) \end{aligned}$$

we have $n = 4$, $p = 2$ and :

$${}^tA = \begin{pmatrix} 20 & -4 & 0 & 2 & 3 \\ 10 & -2 & 1 & 0 & -1 \end{pmatrix}$$

Two matrix multiplications will be of interest in what follows :

- Au, where $u \in \mathbb{R}^p$, represents a linear combination of our p variables, expressed on the ε_i basis,
- ${}^tAe$, where $e \in \mathbb{R}^{n+1}$, $e_0 = 1$ and $\|e\|_\infty = \max_{0 \le i \le n} |e_i| \le 1$ represents the vector of actual values that our p variables take

Two matrix multiplications will be of interest in what follows :

- Au, where $u \in \mathbb{R}^p$, represents a linear combination of our p variables, expressed on the ε_i basis,
- ${}^tAe$, where $e \in \mathbb{R}^{n+1}$, $e_0 = 1$ and $\|e\|_\infty = \max_{0 \leq i \leq n} |e_i| \leq 1$ represents the vector of actual values that our p variables take

Concretisation

The concretisation of A is the zonotope

$$\gamma(A) = \left\{ {}^tA^t(1|e) \mid e \in R^n, \|e\|_\infty \leq 1 \right\} \subseteq \mathbb{R}^p.$$

We call its linear concretisation the zonotope centered on 0

$$\gamma_{lin}(A) = \left\{ {}^tAe \mid e \in \mathbb{R}^{n+1}, \|e\|_\infty \leq 1 \right\} \subseteq \mathbb{R}^p.$$

Two kinds of noise symbols

- Input noise symbols (ε_i): created by uncertain inputs
- Perturbation noise symbols (η_j): created by uncertainty in analysis

Perturbed affine sets $X = (C^X, P^X)$

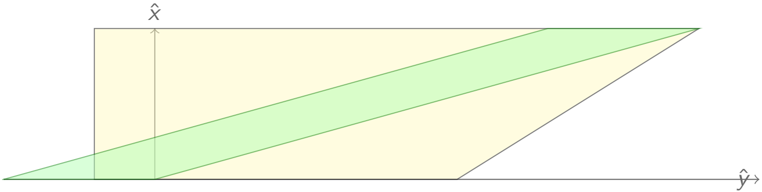
$$\begin{pmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \dots \\ \hat{x}_p \end{pmatrix} = {}^tC^X \begin{pmatrix} 1 \\ \varepsilon_1 \\ \dots \\ \varepsilon_n \end{pmatrix} + {}^tP^X \begin{pmatrix} \eta_1 \\ \eta_2 \\ \dots \\ \eta_m \end{pmatrix}$$

- Central part links the current values of the program variables to the initial values of the input variables (linear functional)
- Perturbation part encodes the uncertainty in the description of values of program variables due to non-linear computations (multiplication, join etc.)

Zonotopes define input-output relations (parameterization by the ε_i)

- Want an order that preserves these input-output relations

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



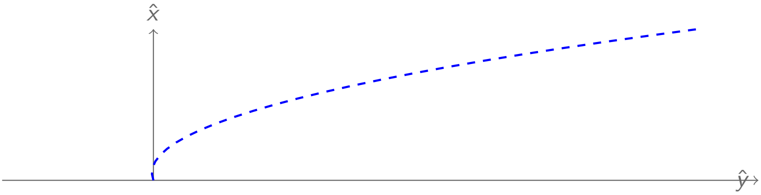
Zonotope (green) is

$$\begin{aligned} x &= 5 + 5\varepsilon_1 \\ y &= (5 + 5\varepsilon_1)(5 + 5\varepsilon_1) - 5 - 5\varepsilon_1 \\ &= 20 + 45\varepsilon_1 + 25\varepsilon_1^2 = 20 + 45\varepsilon_1 + 25\,(0.5 + 0.5\eta_1) \\ &= 32.5 + 45\varepsilon_1 + 12.5\eta_1 \end{aligned}$$

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

Functional interpretation of this example

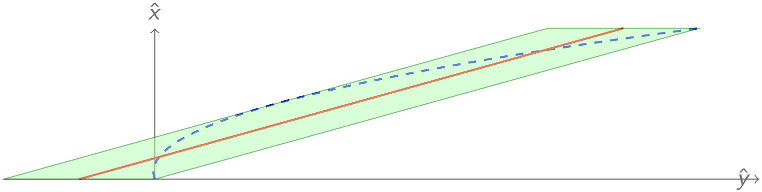
```
real x = [0,10];
real y = x*x - x;
```



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

$$y = 32.5 + 45\varepsilon_1 + 12.5\eta_1$$

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



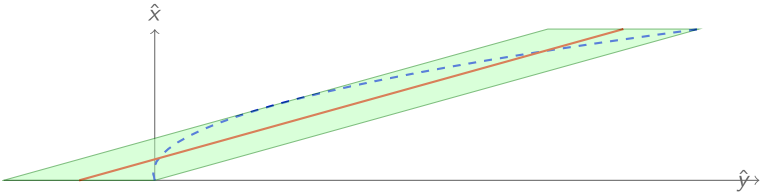
Abstraction of function $x \to y = x^2 - x$ as

$$\begin{aligned} y &= 32.5 + 45\varepsilon_1 + 12.5\eta_1 \\ &= -12.5 + 9x + 12.5\eta_1 \end{aligned}$$

(since $x = 5 + 5\varepsilon_1$)

Functional interpretation of this example

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

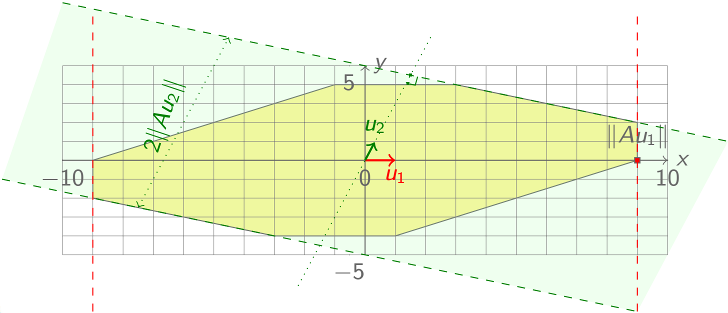


Abstraction of function $x \to y = x^2 - x$ as

$$\begin{aligned} y &= 32.5 + 45\varepsilon_1 + 12.5\eta_1 \\ &= -12.5 + 9x + 12.5\eta_1 \end{aligned}$$

- Usual order relation on sub-polyhedric abstract domains: geometric ordering
- This will be almost the case here: functional order will be geometric ordering on an augmented zonotope
- Geometric ordering on zonotopes: for zonotopes X, Y centered at 0, $X \subseteq Y$ if and only if for all $u \in \mathbb{R}^p$

$$\|Xu\|_1 \leq \|Yu\|_1 \text{ (where } \|(\alpha_0, \ldots, \alpha_n)\|_1 = \sum_{i=0}^{n} |\alpha_i| \text{)}$$



Concretization in terms of sets of functions from $\mathbb{R}^n$ to $\mathbb{R}^p$:

$$\gamma_f(X) = \left\{ f : \mathbb{R}^n \to \mathbb{R}^p \mid \forall \epsilon \in [-1,1]^n, \exists \eta \in [-1,1]^m, f(\varepsilon) = {}^tC^X \begin{pmatrix} 1 \\ \varepsilon \end{pmatrix} + {}^tP^X \eta \right\}.$$

- Equivalent to the geometric ordering on augmented space $\mathbb{R}^{p+n}$: zonotopes enclosing current values of variables + their initial values ε_i

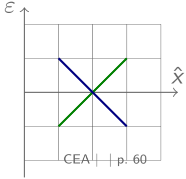
$$\gamma(\tilde{X}) \subseteq \gamma(\tilde{Y}), \text{ where } \tilde{X} = \begin{pmatrix} \begin{pmatrix} 0_n \\ I_{n \times n} \end{pmatrix} & C^X \\ 0_{m \times n} & P^X \end{pmatrix}$$

  - Implies the geometric ordering in $\mathbb{R}^p$
- Computable inclusion test:

$$X \sqsubseteq Y \iff \sup_{u \in \mathbb{R}^p} \left(\|(C^Y - C^X)u\|_1 + \|P^X u\|_1 - \|P^Y u\|_1 \right) \leq 0$$

Example

- $x_1 = 2 + \varepsilon_1$, $x_2 = 2 - \varepsilon_1$
- x_1 and x_2 are incomparable



Functional order relation

Concretization in terms of sets of functions from $\mathbb{R}^n$ to $\mathbb{R}^p$:

$$\gamma_f(X) = \left\{ f : \mathbb{R}^n \to \mathbb{R}^p \mid \forall \epsilon \in [-1,1]^n, \exists \eta \in [-1,1]^m, f(\varepsilon) = {}^tC^X \begin{pmatrix} 1 \\ \varepsilon \end{pmatrix} + {}^tP^X \eta \right\} .$$

- Equivalent to the geometric ordering on augmented space $\mathbb{R}^{p+n}$: zonotopes enclosing current values of variables + their initial values ε_i

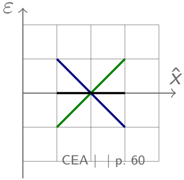
$$\gamma(\tilde{X}) \subseteq \gamma(\tilde{Y}), \text{ where } \tilde{X} = \begin{pmatrix} \begin{pmatrix} 0_n \\ I_{n\times n} \end{pmatrix} & C^X \\ 0_{m\times n} & P^X \end{pmatrix}$$

  - Implies the geometric ordering in $\mathbb{R}^p$
- Computable inclusion test:

$$X \sqsubseteq Y \iff \sup_{u \in \mathbb{R}^p} \left(\|(C^Y - C^X)u\|_1 + \|P^X u\|_1 - \|P^Y u\|_1 \right) \leq 0$$

Example

- $x_1 = 2 + \varepsilon_1$, $x_2 = 2 - \varepsilon_1$ (geometric concretization $[1,3]$)
- x_1 and x_2 are incomparable



Concretization in terms of sets of functions from $\mathbb{R}^n$ to $\mathbb{R}^p$:

$$\gamma_f(X) = \left\{ f : \mathbb{R}^n \to \mathbb{R}^p \mid \forall \epsilon \in [-1,1]^n, \exists \eta \in [-1,1]^m, f(\varepsilon) = {}^tC^X \begin{pmatrix} 1 \\ \varepsilon \end{pmatrix} + {}^tP^X \eta \right\}.$$

- Equivalent to the geometric ordering on augmented space $\mathbb{R}^{p+n}$: zonotopes enclosing current values of variables + their initial values ε_i

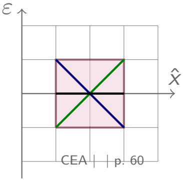
$$\gamma(\tilde{X}) \subseteq \gamma(\tilde{Y}), \text{ where } \tilde{X} = \begin{pmatrix} \begin{pmatrix} 0_n \\ I_{n\times n} \end{pmatrix} & C^X \\ 0_{m\times n} & P^X \end{pmatrix}$$

  - Implies the geometric ordering in $\mathbb{R}^p$
- Computable inclusion test:

$$X \sqsubseteq Y \iff \sup_{u \in \mathbb{R}^p} \left(\|(C^Y - C^X)u\|_1 + \|P^X u\|_1 - \|P^Y u\|_1 \right) \leq 0$$

Example

- $x_1 = 2 + \varepsilon_1$, $x_2 = 2 - \varepsilon_1$, $x_3 = 2 + \eta_1$ (geometric concretization $[1,3]$)
- x_1 and x_2 are incomparable , both are included in x_3.



Addition

Let $X = (C^X, P^X)$ be a form in $\mathcal{M}(n+1, p) \times \mathcal{M}(m, p)$. We define $Z = [\![x_{p+1} = x_i + x_j]\!]X = (C^Z, P^Z) \in \mathcal{M}(n+1, p+1) \times \mathcal{M}(m, p+1)$ by

$$C^Z = \left(\begin{array}{c|c} C^X & \begin{array}{c} c^X_{0,i} + c^X_{0,j} \\ \dots \\ c^X_{n,i} + c^X_{n,j} \end{array} \end{array} \right) \text{ and } P^Z = \left(\begin{array}{c|c} P^X & \begin{array}{c} p^X_{1,i} + p^X_{1,j} \\ \dots \\ p^X_{m,i} + p^X_{m,j} \end{array} \end{array} \right).$$

Abstract interpretation at work: is this a sound over-approximation?

- Monotony of the abstract addition wrt the order on affine sets
- Over-approximates the concrete behaviours

Proof of monotony

Remember the inclusion test:

$$X \sqsubseteq Y \iff \sup_{u \in \mathbb{R}^p} \left(\|(C^Y - C^X)u\|_1 + \|P^X u\|_1 - \|P^Y u\|_1 \right) \leq 0$$

Monotony: we want to prove

$$X \sqsubseteq Y \Rightarrow [\![x_{p+1} = x_i + x_j]\!]X \sqsubseteq [\![x_{p+1} = x_i + x_j]\!]Y$$

Proof. For all $t \in \mathbb{R}^{p+1}$:

$$\begin{aligned}
\|(C^{[\![x_{p+1}=x_i+x_j]\!]X} &- C^{[\![x_{p+1}=x_i+x_j]\!]Y})t\|_1 = \\
&= \sum_{l=0}^{n} \mid \sum_{k=1}^{p+1} (c_{l,k}^{[\![x_{p+1}=x_i+x_j]\!]X} - c_{l,k}^{[\![x_{p+1}=x_i+x_j]\!]Y})t_k \mid \\
&= \sum_{l=0}^{n} \mid \sum_{k=1}^{p} (c_{l,k}^X - c_{l,k}^Y)t_k + (c_{i,k}^X + c_{j,k}^X)t_{p+1} \mid \\
&= \|(C^X - C^Y)^t(t_1, \ldots, t_i + t_{p+1}, \ldots, t_j + t_{p+1}, \ldots, t_p)\| \\
&\leq \|P^{Y\,t}(t_1, \ldots, t_i + t_{p+1}, \ldots, t_j + t_{p+1}, \ldots, t_p)\| \\
&\quad -\|P^{X\,t}(t_1, \ldots, t_i + t_{p+1}, \ldots, t_j + t_{p+1}, \ldots, t_p)\| \\
&= \|P^{[\![x_{p+1}=x_i+x_j]\!]Y} t\| - \|P^{[\![x_{p+1}=x_i+x_j]\!]X} t\|
\end{aligned}$$

And the concretization is OK and even exact!

Remember the concretization:

$$\gamma_f(X) = \left\{ f : \mathbb{R}^n \to \mathbb{R}^p \mid \forall \varepsilon \in [-1,1]^n, \exists \eta \in [-1,1]^m, f(\varepsilon) = {}^tC^X \begin{pmatrix} 1 \\ \varepsilon \end{pmatrix} + {}^tP^X \eta \right\}.$$

Proof.

$$\begin{array}{rcl}
\gamma_f\left(\llbracket x_{p+1} = x_i + x_j \rrbracket X\right) & = & \{ f : \mathbb{R}^n \to \mathbb{R}^{p+1} \mid \forall \varepsilon \in [-1,1]^n, \exists \eta \in [-1,1]^m, \\
& & f(\varepsilon) = {}^tC^{\llbracket x_{p+1}=x_i+x_j \rrbracket X\, t}(1|\varepsilon) + {}^tP^{\llbracket x_{p+1}=x_i+x_j \rrbracket X\, t}(\eta) \\
& = & \{ (f_1, \ldots, f_{p+1}) \mid \\
& & \left\{ \begin{array}{l} f_u(\varepsilon) = c^X_{0,u} + \sum_{l=1}^n c_{l,u}\varepsilon_l + \sum_{l=1}^m p_{l,u}\eta_l, \;\; u = 1, \ldots, p \\ f_{p+1}(\varepsilon) = c_{0,i} + c_{0,j} + \sum_{l=1}^n (c_{l,i} + c_{l,j})\varepsilon_l + \\ \sum_{l=1}^m (p_{l,i} + p_{l,j})\eta_l \end{array} \right. \\
& = & \{ (f_1, \ldots, f_p, f_{p+1}) \mid (f_1, \ldots, f_p) \in X, f_{p+1} = f_i + f_j \}
\end{array}$$

More complicated for multiplication (abstraction not exact)!

Not a lattice

- No least upper bound of 2 zonotopes, possibly several incomparable minimal upper bounds
- Join operator: we want efficient algorithms to compute upper bounds, if possible minimal ones
- Interpretation of test condition: we need an over-approximation of all lower bounds
  - we will introduce constrained zonotopes (geometric concretization may no longer be a zonotope)

The Kleene iteration will rely on the particular join operator we choose

The choice of "home-made" functional join and meet operations

- Keep the parameterization by the ε_i
- These operations should not be expensive

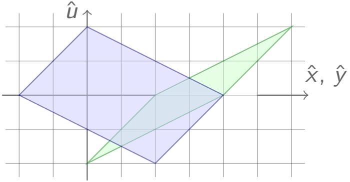
A lot of litterature on zonotopes

- Control theory and hybrid systems analysis: same problem of intersection of zonotopes with guards (Girard, Le Guernic etc)
- But these methods are geometrical
- Still, could be used on the perturbation part

Now: our join operator (2006-present, with E. Goubault)

- Join on coefficients of the forms (interval coefficients): no!
- Central form plus deviation (SAS 2006): $\gamma(\hat{x} \cup \hat{y})$ in general larger than $\gamma(\hat{x}) \cup \gamma(\hat{y})$, bad for fixpoint computation
- Arxiv 2008 and 2009: the componentwise upper bound presented here

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



[demo ex_union.c]

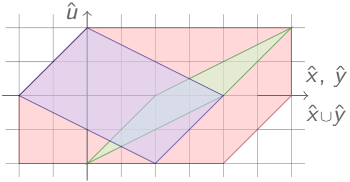
Construction (cost $\mathcal{O}(n \times p)$)

- Keep "minimal common dependencies"

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

Component-wise upper bound (mub under some conditions)

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



[demo ex_union.c]

Construction (cost $\mathcal{O}(n \times p)$)

- 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: $\gamma(\hat{x} \cup \hat{y}) = \gamma(\hat{x}) \cup \gamma(\hat{y})$

Fixpoint computation with componentwise join (arxiv 2008)

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

$$x_{k+n+1} = \sum_{i=1}^{n} a_i x_{k+i} + \sum_{j=1}^{n+1} b_j e_{k+j}, \quad e[*] = input(m, M);$$

- Suppose this concrete scheme has bounded outputs (zeros of $x^n - \sum_{i=0}^{n-1} a_{i+1} x^i$ have modules stricty lower than 1).
- Then there exists q such that the Kleene abstract scheme "unfolded modulo 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, potentially with a widening partly losing dependency information
  - The abstract scheme is a perturbation (by the join operation) of the concrete scheme
  - Uses the stability property of our join operator: for each dimension $\gamma(\hat{x} \cup \hat{y}) = \gamma(\hat{x}) \cup \gamma(\hat{y})$

Some results with the APRON library (K. Ghorbal's Taylor1+ abstract domain, CAV 2009)

without widening (for $p = 5$ and $p = 16$)

filter o2	fixpoint	t(s)
Boxes	$\top$	0.12
Octagons	$\top$	2.4
Polyhedra	$[-1.3,2.82]$	0.53
T.1+(5)	$[-8.9,10.6]$	0.18
T.1+(16)	$[-5.3,6.95]$	0.13

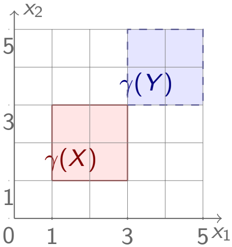
filter o8	fixpoint	t(s)
Boxes	$\top$	0.41
Octagons	$\top$	450
Polyhedra	abort	$>24h$
T.1+(5)	$\top$	360
T.1+(16)	$\top$	942

with widening after 10 Kleene iterations (for $p = 5$ and $p = 20$)

filter o2	fixpoint	t(s)
Boxes	$\top$	<0.01
Octagons	$\top$	0.02
Polyhedra	$\top$	0.59
T.1+(5)	$[-8.9,10.6]$	0.1
T.1+(20)	$[-5.4,7.07]$	0.2

filter o8	fixpoint	t(s)
Boxes	$\top$	<0.01
Octagons	$\top$	2.56
Polyhedra	abort	$>24h$
T.1+(5)	$[-8.9,10.6]$	0.1
T.1+(20)	$[-5.4,7.07]$	0.2

```
real x1 := [1,3];
real x2 := [1,3];
if (random()) {
  x1 = x1 + 2;
  x2 = x2 + 2; }
```

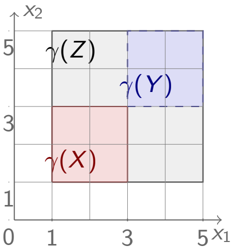


- Joining the two branches: join X and Y defined by

$$X = \begin{pmatrix} \hat{x}_1 = 2 + \varepsilon_1 \\ \hat{x}_2 = 2 + \varepsilon_2 \end{pmatrix} \text{ and } Y = \begin{pmatrix} \hat{y}_1 = 4 + \varepsilon_1 \\ \hat{y}_2 = 4 + \varepsilon_2 \end{pmatrix}$$

Improved join operator: motivation

```
real x1 := [1,3];
real x2 := [1,3];
if (random()) {
  x1 = x1 + 2;
  x2 = x2 + 2; }
```



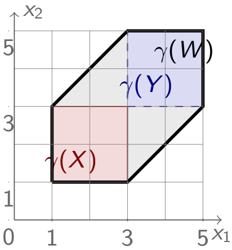
- Joining the two branches: join X and Y defined by

$$X = \begin{pmatrix} \hat{x}_1 = 2 + \varepsilon_1 \\ \hat{x}_2 = 2 + \varepsilon_2 \end{pmatrix} \text{ and } Y = \begin{pmatrix} \hat{y}_1 = 4 + \varepsilon_1 \\ \hat{y}_2 = 4 + \varepsilon_2 \end{pmatrix}$$

- Component-wise join:

$$Z = \begin{pmatrix} \hat{z}_1 = 3 + \varepsilon_1 + \eta_1 \\ \hat{z}_2 = 3 + \varepsilon_2 + \eta_2 \end{pmatrix}$$

```
real x1 := [1,3];
real x2 := [1,3];
if (random()) {
  x1 = x1 + 2;
  x2 = x2 + 2; }
```



- Joining the two branches: join X and Y defined by

$$X = \begin{pmatrix} \hat{x}_1 = 2 + \varepsilon_1 \\ \hat{x}_2 = 2 + \varepsilon_2 \end{pmatrix} \text{ and } Y = \begin{pmatrix} \hat{y}_1 = 4 + \varepsilon_1 \\ \hat{y}_2 = 4 + \varepsilon_2 \end{pmatrix}$$

- Component-wise join:

$$Z = \begin{pmatrix} \hat{z}_1 = 3 + \varepsilon_1 + \eta_1 \\ \hat{z}_2 = 3 + \varepsilon_2 + \eta_2 \end{pmatrix}$$

- Relation between variables and inputs of the program true for both branches joined: $x_2 - x_1 = \varepsilon_2 - \varepsilon_1$
  - use join on one variable $W_1 = 3 + \varepsilon_1 + \eta_1$, and deduce the other by the relation: $W_2 = 3 + \varepsilon_2 + \eta_1$.

Improved join (NSAD 2012, with T. Le Gall, E. Goubault)

Compute and preserve relations between variables and noise symbols

- Compute $k < p$ independent affine relations common to the 2 abstract values joined (solving a linear system)
- Componentwise join on $p - k$ components
- Global join for the k remaining components using the relations
- mub on the $p - k$ components $\Rightarrow$ mub on all components (else ub)

Application: exhibits some implicit relations

```
real x=[0,4];
int i=0;
while (i <= 5) {
    i++;
    x++;
}
```

Relation $x - i = 2 + 2\varepsilon_1$:
$i = 3 + 3\eta_2 \in [0, 6]$,
$x = 5 + 2\varepsilon_1 + 3\eta_2 \in [0, 10]$

A difficulty: "imprecise" relations (fp computations etc)

Main idea to interpret test, informally

- 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)
- More general constraints in the future ?

Example

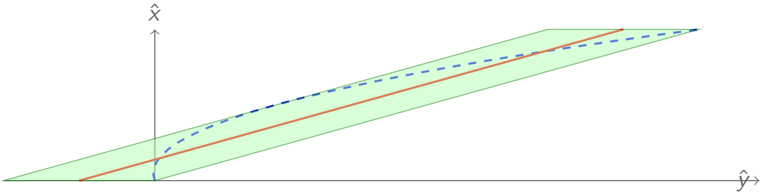
```
real x = [0,10];
real y = 2*x;
if (y >= 10)
  y = x;
```

- 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

(Minimal) upper bound computation on constrained affine sets is difficult

Remember: example of functional abstraction

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



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

$$\begin{aligned} y &= 32.5 + 45\varepsilon_1 + 12.5\eta_1 \\ &= -12.5 + 9x + 12.5\eta_1 \end{aligned}$$

- Almost modular by construction!
- But valid only for inputs in [0,10] $\Rightarrow$ partition of contexts though a summary-based algorithm.

A summary of a program function is a pair of zonotopes (I, O)

- I abstracts the calling context, O the output
- Both zonotopes abstract functions of the inputs of the program (linear form of the ε_i)
- Output O can be instantiated for a particular calling context $C \subseteq I$ (constraints on the noise symbols that define a restriction of the function)

Summary instantiation: $[\![I == C]\!]O$

- Write the constraint $I == C$ in the space of noise symbols:

$$(c^I_{0i} - c^C_{0i}) + \sum_{r=1}^{n} (c^I_{ri} - c^C_{ri})\varepsilon_r + \sum_{r=1}^{m} (p^I_{ri} - p^C_{ri})\eta_r = 0 \ (i = 1, \ldots, p)$$

- We derive relations of the form (by Gauss elimination):

$$\eta_{k_{i+1}} = R_i(\eta_{k_i}, \ldots, \eta_1, \varepsilon_n, \ldots, \varepsilon_1) \ (i = 0, \ldots, r-1)$$

- We eliminate $\eta_{k_1}, \ldots, \eta_{k_r}$ in O using the relations above

Example

```
(0) mult(real a, real b) { return a*(b-2); }
(1) x := [-1,1];  // x = eps_1
(2) real y1 = mult(x+1, x);
(3) real y2 = mult(x, 2*x);
```

if $!(C \leq I)$ **then**
$I \leftarrow I \sqcup C$
$S_f \leftarrow (I, [\![f]\!](I))$
end if
return $[\![I == C]\!]O$

$S_f = \emptyset$
$(2) : I_1 = C_1 = (\varepsilon_1 + 1, \varepsilon_1)$

Example

```
(0) mult(real a, real b) { return a*(b-2); }
(1) x := [-1,1];   // x = eps_1
(2) real y1 = mult(x+1, x);
(3) real y2 = mult(x, 2*x);
```

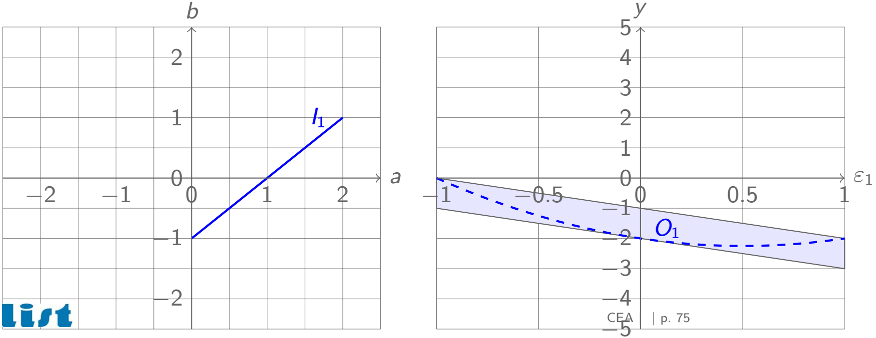
if $!(C \leq I)$ **then**
  $I \leftarrow I \sqcup C$
  $S_f \leftarrow (I, [\![f]\!](I))$
end if
return $[\![I == C]\!]O$

$(2) : I_1 = C_1 = (\varepsilon_1 + 1, \varepsilon_1)$
$O_1 = [\![mult]\!](I_1) = -1.5 - \varepsilon_1 + 0.5\eta_1$
$S = (I_1, O_1)$

```
(2) real y1 = mult(x+1, x);
(3) real y2 = mult(x, 2*x);
```

if $!(C \leq I)$ **then**
$I \leftarrow I \sqcup C$
$S_f \leftarrow (I, [\![f]\!](I))$
end if
return $[\![I == C]\!]O$

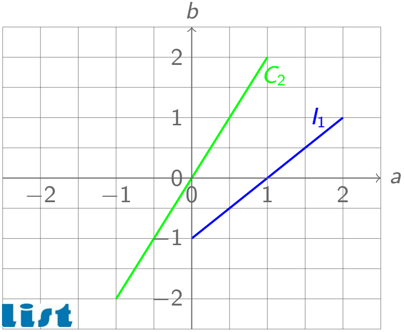
$(2) : I_1 = (\varepsilon_1 + 1, \varepsilon_1)$
$O_1 = (-1.5 - \varepsilon_1 + 0.5\eta_1), S = (I_1, O_1)$

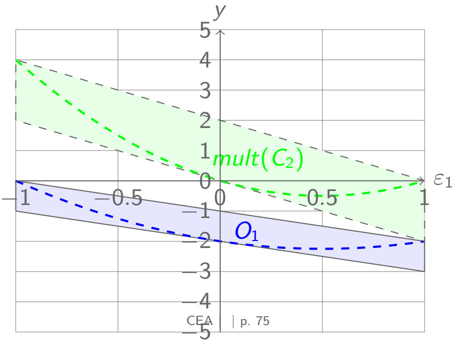


```
(2)  real y1 = mult(x+1, x);
(3)  real y2 = mult(x, 2*x);
```

if $!(C \leq I)$ **then**
$I \leftarrow I \sqcup C$
$S_f \leftarrow (I, [\![f]\!](I))$
end if
return $[\![I == C]\!]O$

$(2): I_1 = (\varepsilon_1 + 1, \varepsilon_1)$
$(3): C_2 = (\varepsilon_1, 2\varepsilon_1)$

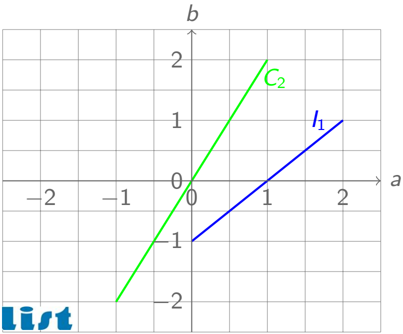


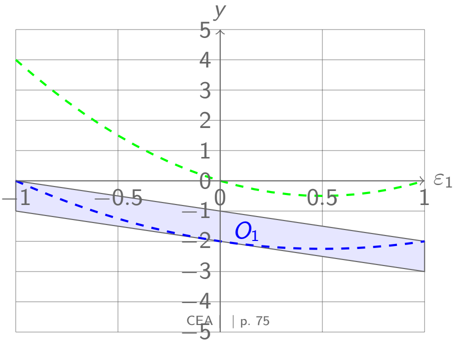


```
(2) real y1 = mult(x+1, x);
(3) real y2 = mult(x, 2*x);
```

if $!(C \leq I)$ **then**
  $I \leftarrow I \sqcup C$
  $S_f \leftarrow (I, [\![f]\!](I))$
end if
return $[\![I == C]\!]O$

$(2) : I_1 = (\varepsilon_1 + 1, \varepsilon_1)$
$(3) : C_2 = (\varepsilon_1, 2\varepsilon_1)$





```
(2) real y1 = mult(x+1, x);
(3) real y2 = mult(x, 2*x);
```

if $!(C \leq I)$ **then**

$I \leftarrow I \sqcup C$

$S_f \leftarrow (I, [\![f]\!](I))$

end if

return $[\![I == C]\!]O$

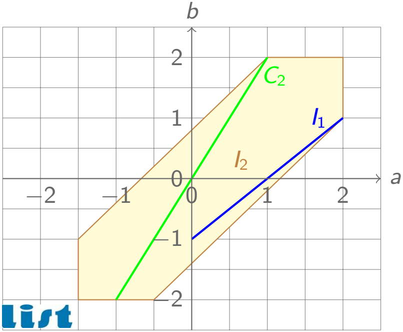
$I_1 = (\varepsilon_1 + 1, \varepsilon_1)$

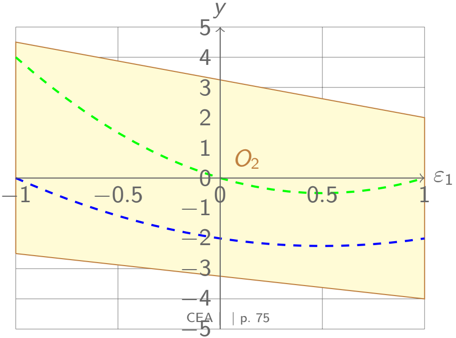
$C_2 = (\varepsilon_1, 2\varepsilon_1)$

$I_2 = I_1 \sqcup C_2 = (\frac{1}{2} + \varepsilon_1 + \frac{1}{2}\eta_2, \frac{3}{2}\varepsilon_1 + \frac{1}{2}\eta_3)$

$O_2 = -\frac{1}{4} - \frac{5}{4}\varepsilon_1 - \eta_2 + \frac{1}{4}\eta_3 + \frac{9}{4}\eta_4$

$S = (I_2, O_2)$





return $[\![I_2 == C_2]\!] O_2$

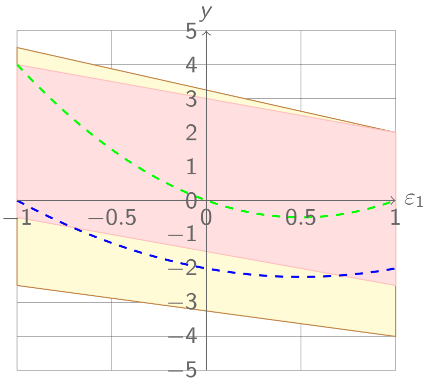
$$C_2 = (\varepsilon_1, 2\varepsilon_1)$$
$$I_2 = (\tfrac{1}{2} + \varepsilon_1 + \tfrac{1}{2}\eta_2, \tfrac{3}{2}\varepsilon_1 + \tfrac{1}{2}\eta_3)$$

$I_2 == C_2$ yields constraints

$$\varepsilon_1 = \tfrac{1}{2} + \varepsilon_1 + \tfrac{1}{2}\eta_2 \Rightarrow \eta_2 = -1$$
$$2\varepsilon_1 = \tfrac{3}{2}\varepsilon_1 + \tfrac{1}{2}\eta_3 \Rightarrow \eta_3 = \varepsilon_1$$

Substitute η_2 and η_3 in
$O_2 = -\frac{1}{4} - \frac{5}{4}\varepsilon_1 - \eta_2 + \frac{1}{4}\eta_3 + \frac{9}{4}\eta_4$
gives

$$[\![I_2 == C_2]\!] O_2 = \frac{3}{4} - \varepsilon_1 + \frac{9}{4}\eta_4.$$



return $[\![I_2 == C_2]\!] O_2$

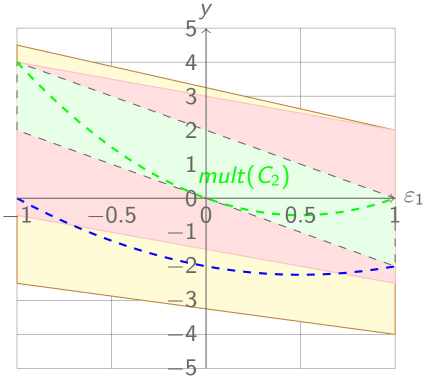
$$C_2 = (\varepsilon_1, 2\varepsilon_1)$$
$$I_2 = (\tfrac{1}{2} + \varepsilon_1 + \tfrac{1}{2}\eta_2, \tfrac{3}{2}\varepsilon_1 + \tfrac{1}{2}\eta_3)$$

$I_2 == C_2$ yields constraints

$$\varepsilon_1 = \tfrac{1}{2} + \varepsilon_1 + \tfrac{1}{2}\eta_2 \Rightarrow \eta_2 = -1$$
$$2\varepsilon_1 = \tfrac{3}{2}\varepsilon_1 + \tfrac{1}{2}\eta_3 \Rightarrow \eta_3 = \varepsilon_1$$

Substitute η_2 and η_3 in
$O_2 = -\frac{1}{4} - \frac{5}{4}\varepsilon_1 - \eta_2 + \frac{1}{4}\eta_3 + \frac{9}{4}\eta_4$
gives

$$[\![I_2 == C_2]\!] O_2 = \frac{3}{4} - \varepsilon_1 + \frac{9}{4}\eta_4.$$



III. Floating-point analysis in FLUCTUAT and case studies

Local outline

- Floating-point computations
- Fluctuat: concrete semantics; academic examples
- Abstraction in Fluctuat: extension of zonotopic domains for finite precision analysis, stable test hypothesis
- Case studies
- Unstable test analysis

- Normalized floating-point numbers

$$(-1)^s 1.x_1x_2\ldots x_n \times 2^e \quad \text{(radix 2 in general)}$$

  - implicit 1 convention ($x_0 = 1$)
  - $n = 23$ for simple precision, $n = 52$ for double precision
  - exponent e is an integer represented on k bits ($k = 8$ for simple precision, $k = 11$ for double precision)
- Denormalized numbers (gradual underflow),

$$(-1)^s 0.x_1x_2\ldots x_n \times 2^{e_{\min}}$$

- Consequences and difficulties:
  - limited range and precision: potentially inaccurate results, run-time errors
  - no associativity, representation error for harmless-looking reals such as 0.1
  - re-ordering by the compiler, use of registers with different precision, etc

Unit in the Last Place (ULP) and IEEE 754 norm (1985) : correct (or exact) rounding

- $ulp(x)$ = distance between two consecutive floating-point numbers around x = maximal rounding error of a number around x
- A few figures for simple precision floating-point numbers :

largest normalized	$\sim$	$3.40282347 * 10^{38}$
smallest positive normalized	$\sim$	$1.17549435 * 10^{-38}$
largest positive denormalized	$\sim$	$1.17549421 * 10^{-38}$
smallest positive denormalized	$\sim$	$1.40129846 * 10^{-45}$
$ulp(1)$	$=$	$2^{-23} \sim 1.19200928955 * 10^{-7}$

- Four rounding modes : to the nearest (default), rounding towards $+\infty$, rounding towards $-\infty$, or rounding towards 0
- The result of $x * y$, ($*$ being $+, -, \times, /$), or of $\sqrt{x}$, is the rounded value (with chosen rounding mode) of the real result
  - the rounding error of such operation is always less than the ulp of the result

Some difficulties of floating-point computation

- Representation error : transcendental numbers π, e, but also

$$\frac{1}{10} = 0.000110011001100110011001100\cdots$$

- Floating-point arithmetic :
  - absorption : $1 + 10^{-8} = 1$ in simple precision float
  - associative law not true : $(-1+1) + 10^{-8} \neq -1 + (1 + 10^{-8})$
  - cancellation : important loss of relative precision when two close numbers are subtracted
- The IEEE 754 norm of 1985 has improved portability but still some non-reproductibility/portability:
  - re-ordering of operations by the compiler
  - storage of intermediate computation either in register or in memory, with different floating-point formats
  - elementary library operators (such as exp/log, trigonometric functions) non specified until 2008, and correct rounding only recommended since 2008

(a, b, c the lengths of the sides of the triangle, a close to $b+c$):

$$A = \sqrt{s(s-a)(s-b)(s-c)} \qquad s = \frac{a+b+c}{2}$$

Then if a,b, or c is known with some imprecision, $s-a$ is very inaccurate. Example,

real number	floating-point number
$a = 1.9999999$	$a = 1.999999881...$
$b = c = 1$	$b = c = 1$
$s - a = 5e - 08$	$s - a = 1.19209e - 07$
$A = 3.16...e - 4$	$A = 4.88...e - 4$

In real world : a catastrophic example

- 25/02/91: a Patriot missile misses a Scud in Dharan and crashes on an american building : 28 deads.
- Cause :
  - 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

$$\frac{1}{10} = 0.0001100110011001100110 \cdots$$

Truncation error	$\sim$	0.000000095 (decimal)
Drift, on 100 hours	$\sim$	$0.34s$
Location error on the scud	$\sim$	$500m$

Some references on floating-point arithmetic

- W. Kahan's web site (father of IEEE 754 norm) http://www.eecs.berkeley.edu/ wkahan/
- Goldberg's article What every computer scientist should know about Floating-Point arithmetic http://www.validlab.com/goldberg/paper.pdf
- The AriC team of LIP (ENS Lyon) http://www.ens-lyon.fr/LIP/AriC/
- The CARAMEL team of LORIA (Nancy) http://caramel.loria.fr

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)

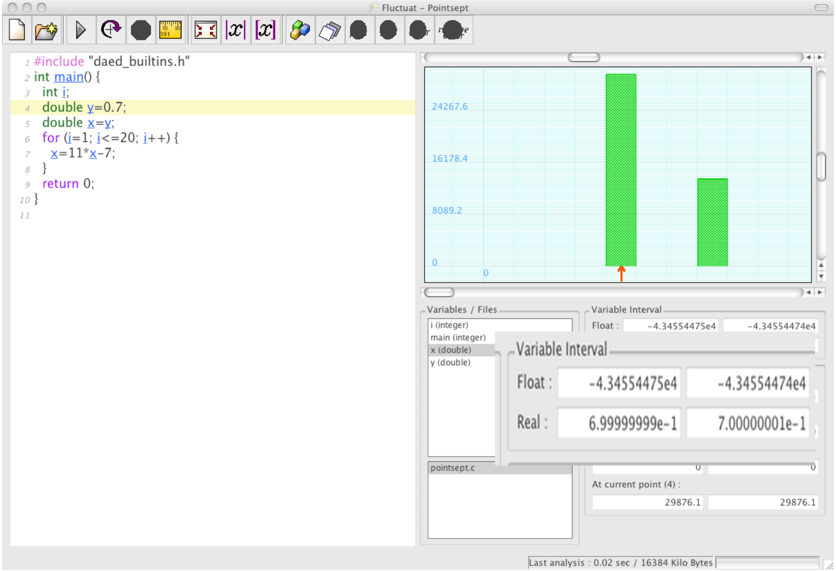
FLUCTUAT: concrete semantics

- 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)

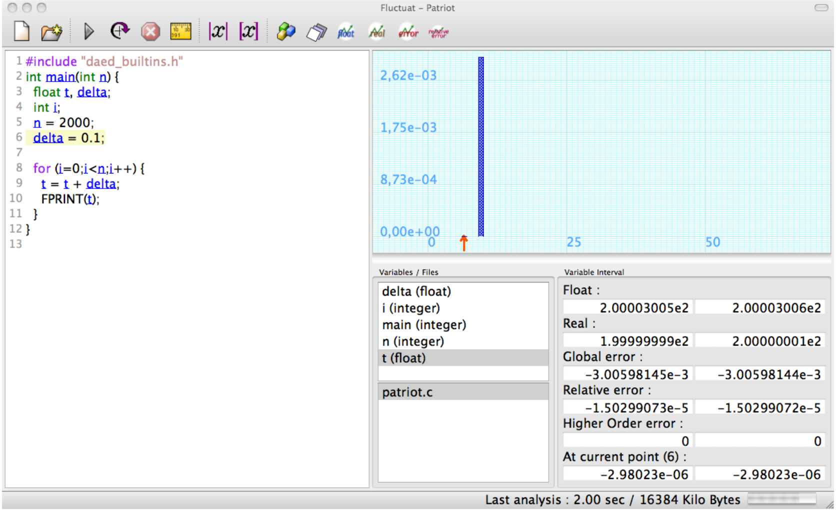
```
float x,y,z;
x = 0.1; // [1]
y = 0.5; // [2]
z = x+y; // [3]
t = x*z; // [4]
```

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

Example (Fluctuat)



Example


Fluctuat – Patriot
#include "daed_builtins.h"
int main(int n) {
float t, delta;
int i;
n = 2000;
delta = 0.1;
for (i=0;i<n;i++) {
t = t + delta;
FPRINT(t);
}
}
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

Compute, with $x_0 = 11/2.0$ and $x_1 = 61/11.0$, the sequence

$$x_{n+2} = 111 - \frac{(1130 - \frac{3000}{x_n})}{x_{n+1}}$$

- Two fixpoint, one attractive, one repulsive:
  - if computed with real numbers, converges to 6. If computed with any approximation, converges to 100.
- Results with Fluctuat :
  - for x_{10} : finds the floating-point value equal to $f_{10} = 100$, with an error e_{10} in [-94.1261,-94.1258], and a real value r_{10} in [5.8812,5.8815]
  - for x_{100} :
    - with default precision of the analysis (fp numbers with 60 bits mantissa), or even 400 mantissa bits numbers, finds $f_{100} = 100$, $e_{100} = \top$ and $r_{100} = \top$: indicates high unstability
    - with 500 mantissa bits numbers, finds $f_{100} = 100$, $e_{100} = -94$ and $r_{100} = 5.99\ldots$

[demo KahanMuller.c]

IEEE 754 norm on f.p. numbers specifies the rounding error:

- 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
  - unit in the Last Place $ulp(x)$ = distance between two consecutive floating-point numbers around x = maximal rounding error around x

$$ulp(1) = 2^{-23} \sim 1.19200928955 * 10^{-7}$$

  - rounding error of elementary operation always less than the ulp of the result
  - also more refined properties, such as Sterbenz lemma (if x and y are two float such that $\frac{y}{2} \leq x \leq y$, then f.p. operation $x - y$ is exact)

Abstraction: for each variable x, a triplet (f^x, r^x, e^x)

Abstract value

- For each variable:
  - 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

Central operation because involved in all arithmetic operations

$$y = (float)^n x \quad = \quad ((float)(\mathbf{f}^x),\ \hat{r}^x,\ \hat{e}^x + new_{\varepsilon_n^e}(\mathbf{e}(\mathbf{f}^x))),$$

- Rounding error of a real/double value given in $\mathbf{f}^x$ to its finite precision representation bounded by the interval

$$\mathbf{e}(\mathbf{f}^x) = [-u^x, u^x] \cap ([\underline{f^x}, \overline{f^x}] - [fl(\underline{f^x}), fl(\overline{f^x})]),$$

  where $u^x = \max(\delta_r \max(|fl(\underline{f^x})|, |fl(\overline{f^x})|), \delta_a)$.
  - computed as the intersection of the bound given by the norm, and the interval difference between the real and the finite precision values.
- Creation of a new error noise symbol ε_n^e associated to control point n: for an interval I, $new_{\varepsilon_n^e}(I) = mid(I) + dev(I)\varepsilon_n^e$

Multiplication

$$z = x \times^n y = ((\mathbf{f}^x \times_{\mathbb{F}} \mathbf{f}^y) \cap \gamma(\hat{r}^z - \hat{e}^z), \hat{r}^x \hat{r}^y, \hat{e}^z),$$

where

$$\hat{e}^z = \hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y + new_{\varepsilon_n^e}\left(\mathbf{e}\left(\gamma\left(\hat{r}^z - \left(\hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y\right)\right)\right)\right)$$

Example:

```
float x = [0,1];      [1]
float y = 0.1;        [2]
float z = x*y;        [3]
```

$$x \quad = \quad ([0,1], 0.5 + 0.5\epsilon_1^r, 0)$$

$$z = x \times^n y = \left((\mathbf{f}^x \times_{\mathbb{F}} \mathbf{f}^y) \cap \gamma(\hat{r}^z - \hat{e}^z), \hat{r}^x \hat{r}^y, \hat{e}^z \right),$$

where

$$\hat{e}^z = \hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y + new_{\varepsilon_n^e} \left(\mathbf{e} \left(\gamma \left(\hat{r}^z - \left(\hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y \right) \right) \right) \right)$$

Example:

```
float x = [0,1];      [1]
float y = 0.1;        [2]
float z = x*y;        [3]
```

$$\begin{aligned} x &= \left([0,1], 0.5 + 0.5\epsilon_1^r, 0 \right) \\ y &= \left(fl(0.1), 0.1, -1.59e^{-9} \right) \end{aligned}$$

Multiplication

$$z = x \times^n y = \left((\mathbf{f}^x \times_{\mathbb{F}} \mathbf{f}^y) \cap \gamma(\hat{r}^z - \hat{e}^z), \hat{r}^x \hat{r}^y, \hat{e}^z \right),$$

where

$$\hat{e}^z = \hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y + new_{\varepsilon_n^e}\left(\mathbf{e}\left(\gamma\left(\hat{r}^z - (\hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y)\right)\right)\right)$$

Example:

```
float x = [0,1];      [1]
float y = 0.1;        [2]
float z = x*y;        [3]
```

$$\begin{aligned}
x &= ([0,1], 0.5 + 0.5\epsilon_1^r, 0) \\
y &= \left(fl(0.1), 0.1, -1.59e^{-9}\right) \\
z &= \left([0, fl(0.1)], 0.05(1 + \varepsilon_1^r), -7.45e^{-10}(1 + \varepsilon_1^r) + 3.72e^{-9}\varepsilon_3^e\right)
\end{aligned}$$

Multiplication

$$z = x \times^n y = \left((\mathbf{f}^x \times_{\mathbb{F}} \mathbf{f}^y) \cap \gamma(\hat{r}^z - \hat{e}^z), \hat{r}^x \hat{r}^y, \hat{e}^z \right),$$

where

$$\hat{e}^z = \hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y + new_{\varepsilon_n^e}\left(\mathbf{e}\left(\gamma\left(\hat{r}^z - (\hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y)\right)\right)\right)$$

Example:

```
float x = [0,1];      [1]
float y = 0.1;        [2]
float z = x*y;        [3]
```

$$\begin{aligned} x &= ([0,1], 0.5 + 0.5\epsilon_1^r, 0) \\ y &= \left(fl(0.1), 0.1, -1.59e^{-9}\right) \\ z &= \left([0, fl(0.1)], 0.05(1+\varepsilon_1^r), \ -7.45e^{-10}(1+\varepsilon_1^r) + 3.72e^{-9}\varepsilon_3^e\right) \end{aligned}$$

Multiplication

$$z = x \times^n y = \left((\mathbf{f}^x \times_{\mathbb{F}} \mathbf{f}^y) \cap \gamma(\hat{r}^z - \hat{e}^z), \hat{r}^x \hat{r}^y, \hat{e}^z \right),$$

where

$$\hat{e}^z = \hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y + new_{\varepsilon_n^e} \left(\mathbf{e} \left(\gamma \left(\hat{r}^z - (\hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y) \right) \right) \right)$$

Example:

```
float x = [0,1];      [1]
float y = 0.1;        [2]
float z = x*y;        [3]
```

$$\begin{aligned} x &= ([0,1], 0.5 + 0.5\epsilon_1^r, 0) \\ y &= (fl(0.1), 0.1, -1.59e^{-9}) \\ z &= ([0, fl(0.1)], 0.05(1+\varepsilon_1^r), -7.45e^{-10}(1+\varepsilon_1^r) + 3.72e^{-9}\varepsilon_3^e) \end{aligned}$$

$$z = x \times^n y = \left((\mathbf{f}^x \times_{\mathbb{F}} \mathbf{f}^y) \cap \gamma(\hat{r}^z - \hat{e}^z), \hat{r}^x \hat{r}^y, \hat{e}^z \right),$$

where

$$\hat{e}^z = \hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y + new_{\varepsilon_n^e} \left(\mathbf{e} \left(\gamma \left(\hat{r}^z - \left(\hat{r}^y \hat{e}^x + \hat{r}^x \hat{e}^y - \hat{e}^x \hat{e}^y \right) \right) \right) \right)$$

Example:

```
float x = [0,1];      [1]
float y = 0.1;        [2]
float z = x*y;        [3]
```

$$\begin{aligned} x &= \left([0,1], 0.5 + 0.5\epsilon_1^r, 0\right) \\ y &= \left(fl(0.1), 0.1, -1.59e^{-9}\right) \\ z &= \left([0, fl(0.1)], 0.05(1+\varepsilon_1^r), -7.45e^{-10}(1+\varepsilon_1^r) + 3.72e^{-9}\varepsilon_3^e\right) \end{aligned}$$

Stable test assumption

- Assumption that the control flow of the program is the same for the finite precision and real values of the program
- The finite precision control flow is followed: in case of unstable test, possibly unsound error bounds
- Later in the presentation: unstable test analysis (more complicated)

Test interpretation

- Affine forms unchanged; extra constraints on noise symbols,
- Two sets of constraints generated by $\hat{r}^x \cap \hat{r}^y$ and $(\hat{r}^x - \hat{e}^x) \cap (\hat{r}^y - \hat{e}^y)$

Join operation

Component-wise, using the join over intervals and affine forms
$x \cup y = (\mathbf{f}^x \cup \mathbf{f}^y, \hat{r}^x \cup \hat{r}^y, \hat{e}^x \cup \hat{e}^y)$.

- Join over affine forms keeps only common relation
- Sources of errors that are not common to all execution paths of the programs will be lost and assigned to the label of the join.

The static analyzer FLUCTUAT

The Fluctuat team

- E. Goubault, S. Putot (2001-), M. Martel (2001-2005, Fluctuat Assembler), F. Védrine (2008-), K. Tekkal (2008-2011, Digiteo OMTE then start-up incubation), T. Le Gall (2012-)
- Continuous support by Airbus and IRSN, more occasional by other users
  - Is/has been used for a wide variety of codes (automotive, nuclear industry, aeronautics, aerospace) of size up to about 50000 LOCs

Assertions in the program analyzed

- Hypotheses on the environment of the program
- Local partitioning and collecting strategies

Exploitation of results

- Warnings: unstable tests, run-time errors
- Identifying problems and refining results: worst case scenarios, symbolic execution, subdivisions
- Library and new interactive version (F. Védrine)

Case study

- Validation of libraries of elementary functions from aeronautic or automotive industry
  - No actual code can be shown but very representative examples below
  - Extract from
    Delmas, Goubault, Putot, Souyris, Tekkal and Védrine, Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software, In FMICS'09
- Types of codes to be analysed
  - C functions or macro-functions: polynomial approximations, interpolators, digital filters, signal integrators, etc.
  - challenges for precise static analysis
    - very large input ranges, to address all operational contexts
    - (implicit) unbounded loop for digital filters
    - sophisticated bitwise operations
      (WCET, determinism & traceability constraints)

```
int i, j;
double f, Var, Const;
...
j=0x43300000; ((int*)(&Const))[0]=j; ((int*)(&Var))[0]=j;
j=0x80000000; ((int*)(&Const))[1]=j; ((int*)(&Var))[1]=j^i;
f = (double)(Var-Const);
```

Results obtained for every basic operator

- Input ranges guaranteeing absence of run-time errors
- Analysis results proving that each operator
  - can introduce only negligible rounding errors
  - cannot significantly amplify errors on inputs
- Compared to the legacy method (intellectual analysis)
  - the analysis process is faster & easier
  - the figures obtained are
    - usually of the same magnitude
    - sometimes much less pessimistic (up to one order of magnitude)
- In some cases, a functional proof of the underlying algorithm is also achieved (in real and floating-point numbers)

- **Linear Filter of order 2**

$$S_i = 0.7E_i - 1.3E_{i-1} + 1.1E_{i-2} + 1.4S_{i-1} - 0.7S_{i-2}$$

where $S_0 = S_1 = 0$, and the E_i are independent inputs in the range $[0,1]$, that can be modelled by

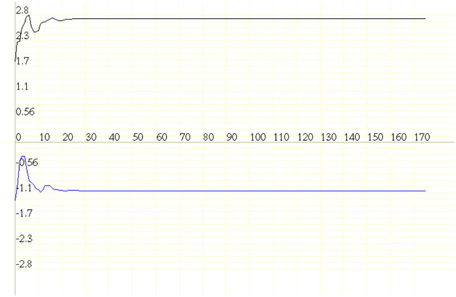
$$\check{E}_i = \hat{E}_i = \frac{1}{2} + \frac{1}{2}\epsilon_i.$$

- **Fixed unfolding** $(i = 99)$:

$$\begin{aligned} \hat{S}_{99} &= \check{S}_{99} = 0.83 + 7.81e^{-9}\varepsilon_0 - 2.1e^{-8}\varepsilon_1 \\ &\quad + \ldots - 0.16\varepsilon_{98} + 0.35\varepsilon_{99} \end{aligned}$$

  - supposing coefficients computed exactly, gives the exact enclosure of S_{99} : $[-1.0907188500, 2.7573854753]$
  - extreme scenario for S_{99} : the coefficients of the affine form allow us to deduce the E_i leading to the max (or min) of the enclosure $(\alpha_i \geq 0 \Rightarrow E_i = 1$ else $E_i = 0)$

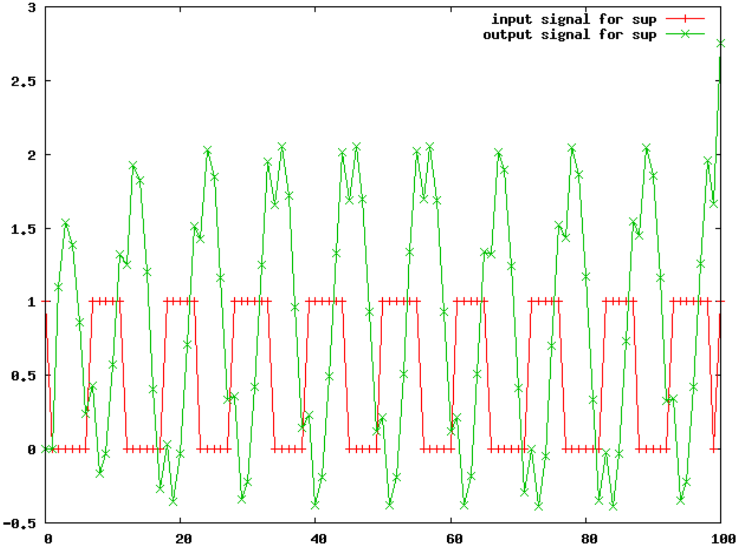
- Concretization of S_i over iterations, in [-1.09,2.76] :

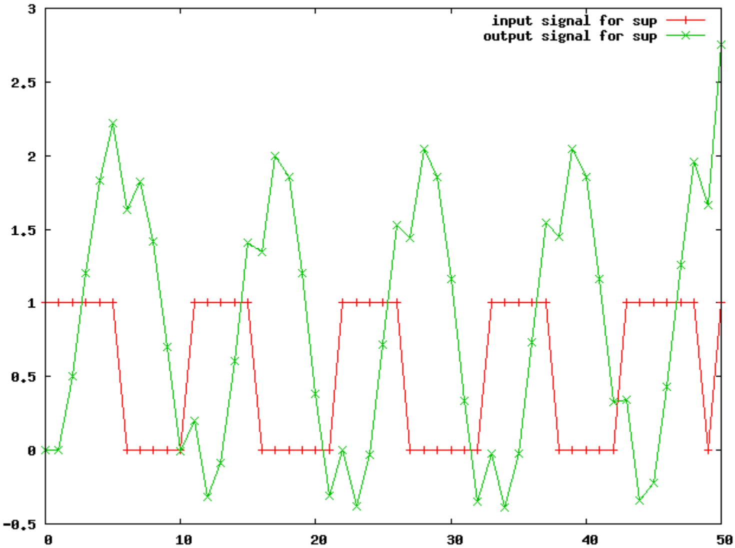


  - for each S_i : over = under-approximation
  - the bounds for the different S_i are not reached with same input sequence

- Real enclosure well approached using finite sequences :
  $S_\infty = [-1.09071884989..., 2.75738551656...]$

The input sequence that maximizes S_{100}





input signal for sup
output signal for sup
3
2.5
2
1.5
1
0.5
0
-0.5
0
10
20
30
40
50

"Worst-case" test generation on the errors: principle

Remember model for real and errors:

$$\hat{r}^x = r_0^x + \sum_i r_i^x \, \varepsilon_i^r$$

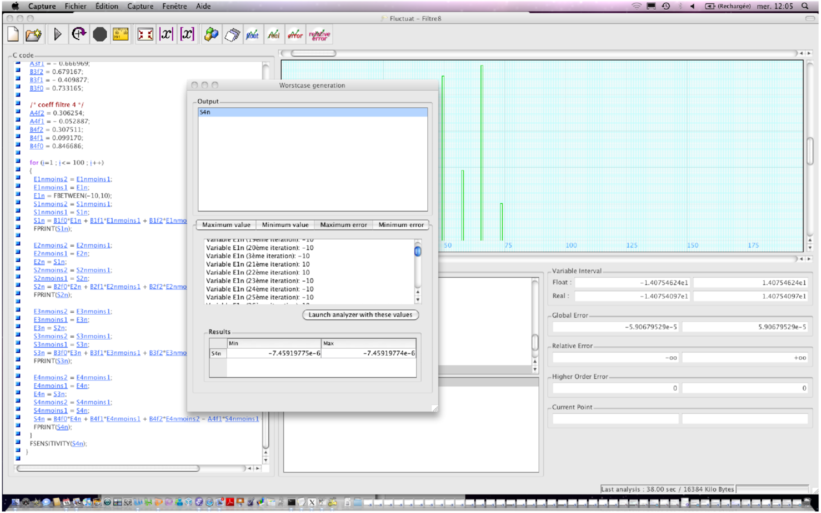
$$\hat{e}^x = e_0^x + \sum_i e_i^x \, \varepsilon_i^r + \sum_l e_l^x \, \varepsilon_l^e$$

- Use e_i^x to find the values that maximize the error, when possible,
- Else $(0 \in e_i^x)$ try to maximize the value,
- Less terms that can be controled compared to worst cases on values but still gives nice results

Example: order 8 filter (4 cascading order 2 filters)

```
  /* coeff 1st filter */
  A1f2 = 0.467388;
  A1f1 = - 1.077138;
...
  for (i=1 ; i<= 100 ; i++) {
    E1nmoins2 = E1nmoins1;    E1nmoins1 = E1n;
    E1n = FBETWEEN(-10,10);
    S1nmoins2 = S1nmoins1;    S1nmoins1 = S1n;
    S1n = B1f0*E1n + B1f1*E1nmoins1 + B1f2*E1nmoins2 - A1f1*S1nmoins1 
    E2nmoins2 = E2nmoins1;    E2nmoins1 = E2n;
...
    S4n = B4f0*E4n + B4f1*E4nmoins1 + B4f2*E4nmoins2 - A4f1*S4nmoins1 
  FSENSITIVITY(S4n);
}
```

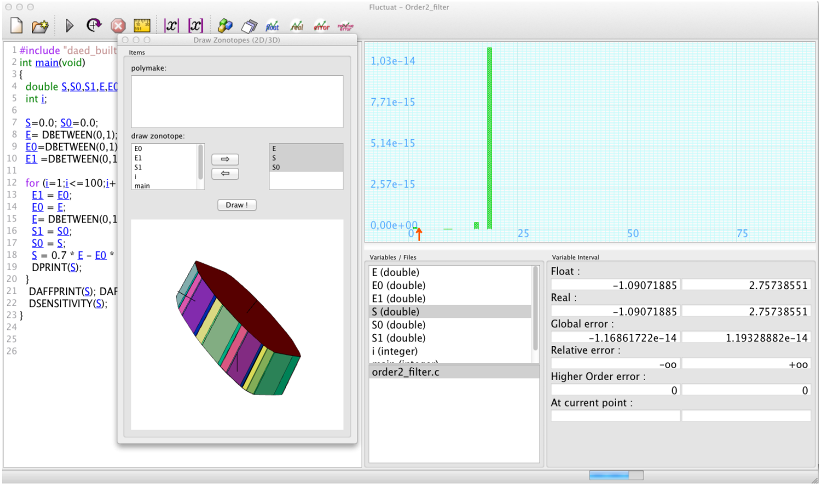
Fluctuat analysis:



[demo order2_filter]

Zonotope (S,S0,E) of the order 2 filter

[demo order2_filter]



Back to the Householder scheme

Study of the conjugate gradient algorithm

Solve $Ax = b$ where matrix A: small perturbation around discretisation of 1D Laplace equation

```
#define N 16             // matrix size
#define epsilon 0.00001
float A[N][N]; float b[N], xi[N], ....;

void evalA(float *x, float *y);                  // y = Ax
float scalarproduct(fl *x, fl *y);               // return <x,y>
void multadd(fl *x,fl *y,fl a,fl b,fl *z);       // z = a*x+b*y

[...]
for (i=0;i<N;i++) {  // init A - discretisation Laplacien 1D
  A[i][i] = FBETWEEN(2.0/(N+1)-0.0000001,2.0/(N+1)+0.0000001);
  if (i < N-1) {
    A[i][i+1] = -1.0/(N+1);
    A[i+1][i] = -1.0/(N+1);
  } }
for (i=0;i<N;i++)  b[i] = 1;
for (i=0;i<N;i++)  xi[i] = FBETWEEN(0,0.0000001);

evalA(xi,temp);                       /* temp = Ax */
multadd(b,temp,1,-1,gi);              /* residue gi = b-Ax */
for (j=0;j<N;j++)  hi[j] = gi[j];  /* descent direction hi = gi */
norm = scalarproduct(gi,gi);          /* residue norm = <gi,gi> */
```

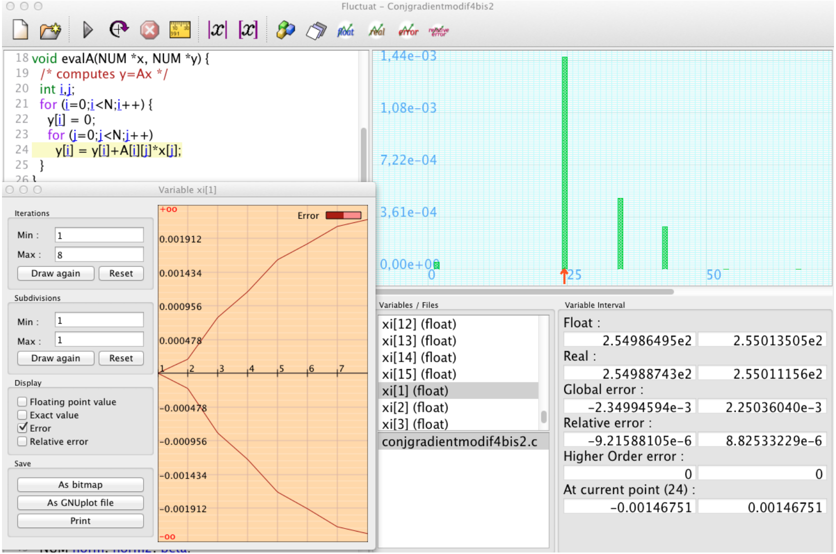
Conjugate gradient algorithm

```
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];
  }
}
```

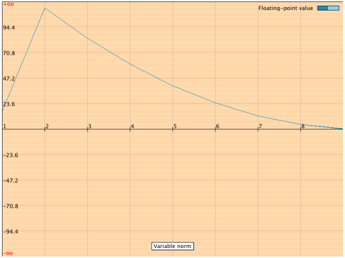
In real numbers: for A symmetric positive definite ($\forall x, < x, Ax >\geq 0$)

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

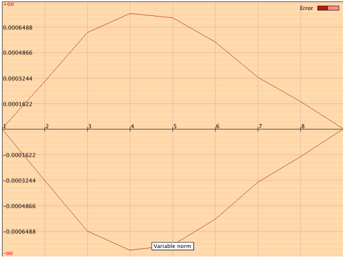
Analysis results: error on one component (x[1])



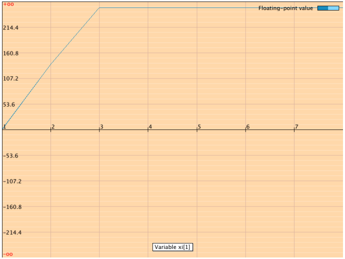
Float and real value of norm are less than $1.e^{-7}$ in 8 iterations



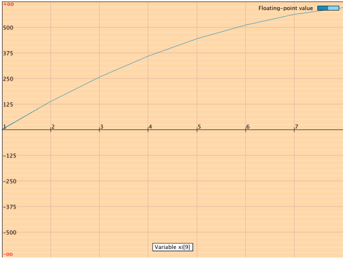
Value of the norm



Error on the norm



Value of `xi[1]`

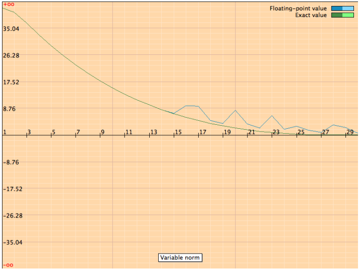


Value of `xi[9]`

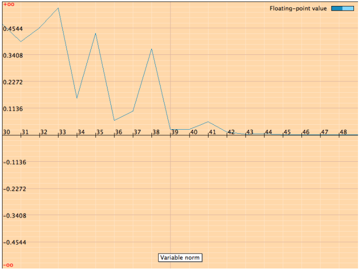
Matrix A is now Strakos matrix in dimension 30

cf The Lanczos and Conjugate Gradient algorithms, from theory to finite precision computations [Meurant 2005]

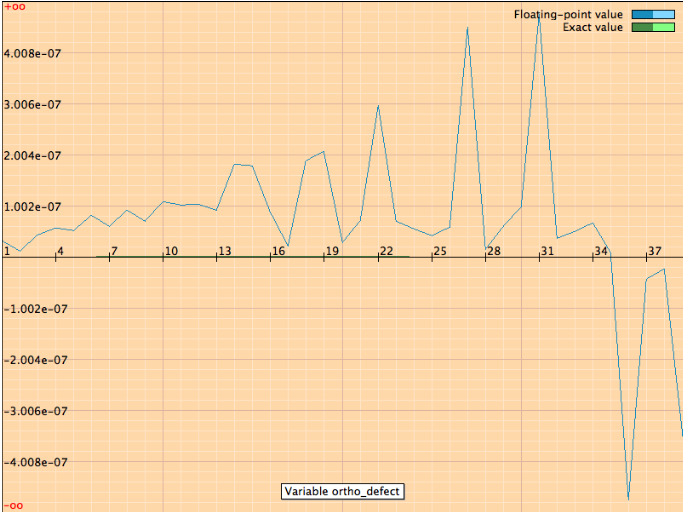
- Condition number around 1000
- Convergence in 30 iterations in real numbers but more difficult in float



Float and real value of the norm



Norm in float for iterates > 30



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

Partial conclusion

We can illustrate classical results on conjugate gradient convergence in finite precision

- On perturbed linear systems with nice behaviour
- On specific matrices known to exhibit difficulties

Try larger classes of problems / perturbations

- Take inspiration from Paige (or successors) error analyses to locally improve the analysis if necessary
- Combination with formal proof methods
- Of course also study self-validating methods for linear systems

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

```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);
  }
}
```

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

$$
\begin{array}{rcl}
\dot{y_0} & = & -y_1 * (y_4 + w12) - y_2 * (y_5 + w22) - y_3 * (y_6 + w32) \\
\dot{y_1} & = & y_0 * (y_4 + w12) + y_2 * (y_6 + w32) - y_3 * (y_5 + w22) \\
\dot{y_2} & = & y_0 * (y_1 + w22) + y_3 * (y_4 + w12) - y_1 * (y_6 + w32) \\
\dot{y_3} & = & y_0 * (y_6 + w32) + y_1 * (y_5 + w22) - y_2 * (y_4 + w12) \\
\dot{y_4} & = & -y_5 * y_6 * i1 + a_0 \\
\dot{y_5} & = & -y_4 * y_6 * i2 + a_1 \\
\dot{y_6} & = & -y_4 * y_5 * i3 + a_2
\end{array}
$$

- 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.

demo: ATV analysis

ATV

Sound unstable test analysis (work in progress)

- Problem of the previous approach: error bounds computed in each branch, may be unsound if a test is unstable
- But when considering large sets of executions, most tests are unstable
- We see now a sound approach, that also computes the discontinuity between branches in conditional blocks if possible unstable test
  - makes our error analysis completely sound
  - also gives a robustness analysis of the implementation

Sound unstable test analysis

- Tests interpreted over real and float values: two sets of constraints on noise symbols
- Joining branches
  - join fp and real values from the branches as previously
  - errors: join error computed in the two branches with, when it exists (unstable test), the difference between real value in one branch and float value in the other branch for the same execution (ie for same values of the ε_i = intersections of the constraints for these two branches)

In the line of robustness/continuity analysis of Chaudhuri, Gulwani and al.

Can small uncertainty on inputs cause only small perturbations on the outputs (with different execution paths): notions classical for control systems but not so much for software implementations.

Some recent work in critical embedded sofware:

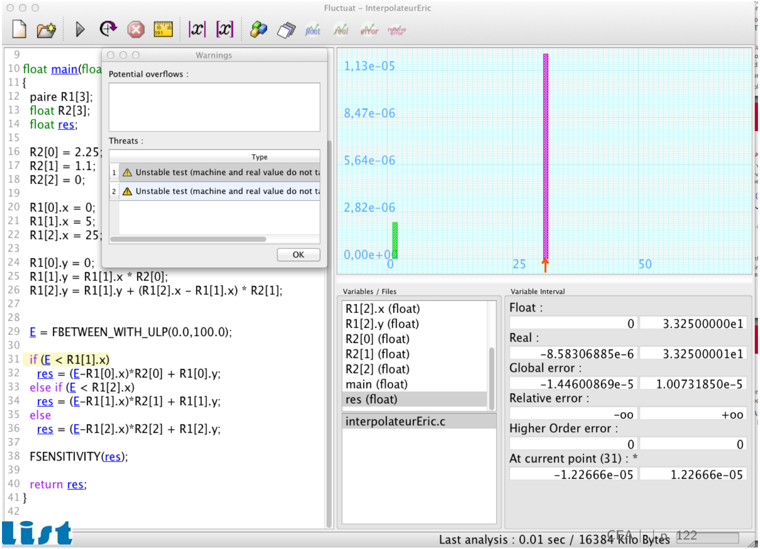
- Some real cases (cf NASA engineer Bushnell's pres. at NSV 2011 on the F22 raptor crossing int. date line in 2007)
- Continuity in Software Systems [Hamlet 2002]
- Continuity analysis and robustness of programs [Chaudhuri, Gulwani, Lublineramn 2010-2012]
- Robust software synthesis, Symbolic robustness analysis, etc [Majumdar, Render, Tabuada, Saha, 2009-2012]

Unstable tests: when real and finite precision control flow can be different

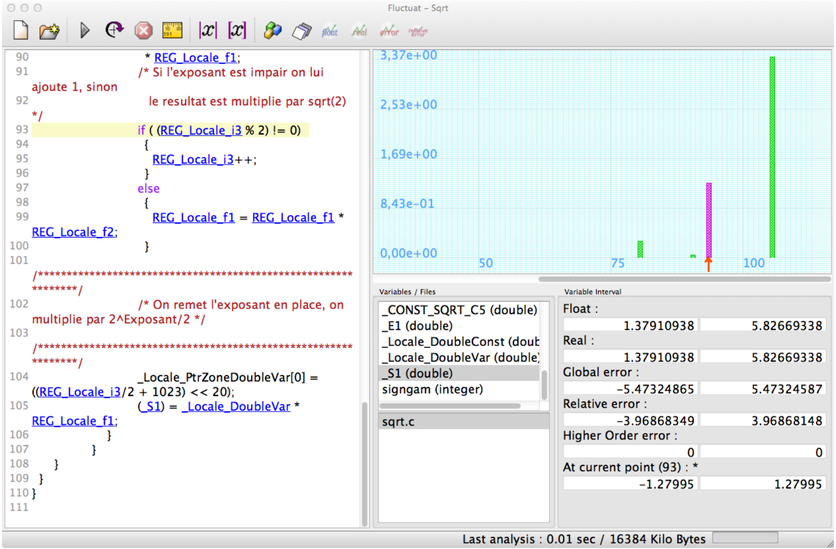
- Error analyses are sound only under the stable test assumption
- When considering large sets of executions, most tests are unstable
- Compute discontinuity error bounds due to unstable tests:
  - makes our error analysis sound in the presence of unstable tests
  - gives a robustness analysis of implementations

A typical example of unstable tests: affine interpolators

All tests are unstable, but the implementation is robust, the conditional block does not introduce a discontinuity



But discontinuities also actually occur (Airbus sqrt)



A similar (modified and simplified) example

```
#define sqrt2 1.4142135381698608398437 50
double x, y;  x = DREAL_WITH_ERROR(1,2,0,0.001);
if (x>2)
  y = sqrt2*(1+(x/2-1)*(.5-0.125*(x/2-1)));
else
  y = 1+(x-1)*(.5+(x-1)*(-.125+(x-1)*.0625));
```

Without unstable test analysis, unsound results in Fluctuat:

- An unstable test is signalled at the `if` statement
- y has real value in [1,1.4531] with an error in [-0.0005312,0.00008592]

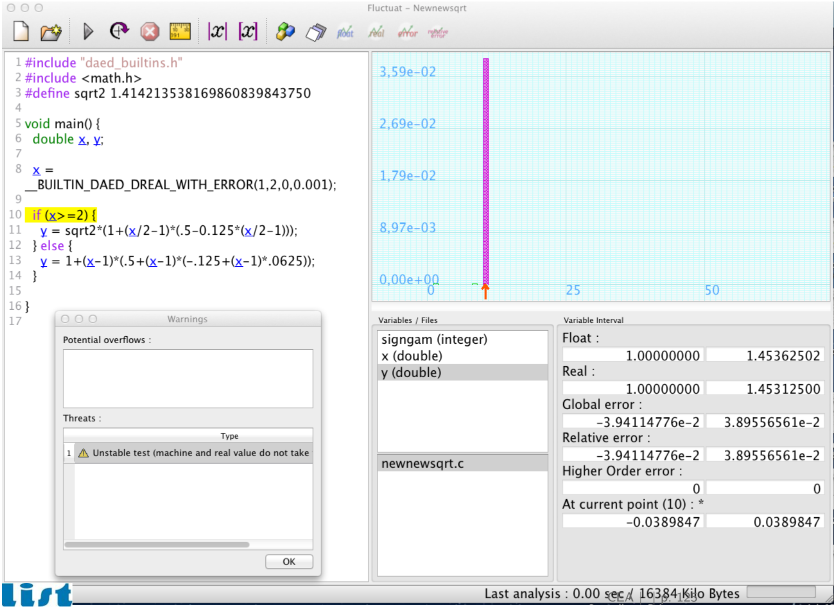
Unstable test: consider for instance $r^x = 2$ and $f^x = 2 + 0.001$

- execution in reals ($r^x = 2$) takes the `else` branch: $r^y = 1.4375$,
- execution in floats ($f^x = 2 + 0.001$) takes the `then` branch: $f^y = 1.4145\ldots$

The test introduces a discontinuity $f^y - r^y = -0.023$ around the test condition ($x == 2$): larger than the error bounds

- want to consider discontinuity as a new error term; accurate abstraction ?

With unstable test analysis



Sound unstable test analysis in Fluctuat

- For each variable, affine forms for real and error:

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

- Constraints on (shared) noise symbols (interpretation of test condition independently over real and float values)
  - for real control flow (test interpretation on r^x: constraints on the ε_i^r)
  - for finite precision control flow (test interpretation on $f^x = r^x + e^x$: constraints on the ε_i^r and ε_l^e)

Join

- On values (float/real): same as before
- On errors: join error computed in each branch, with difference, for unstable tests, between real value in one branch and float value in the other branch
  - unstable test condition: when for a same execution (same values of the noise symbols ε_i) the control flow is different
  - computed as an intersection of constraints on the ε_i: allows us to bound accurately the discontinuity error

Example: sound unstable test analysis

```
int main(void) {
 double x,y;
 x = DREAL_WITH_ERROR(1,3,1.0e-5,1.0e-5);
 if (x <= 2)
   y = x + 2; [1]
 else
   y = x; [2]
}
```

- Before the test: $f^x = (2+\varepsilon_1) + 10^{-5}$
- Test $x \leq 2$:
  - in reals: $\varepsilon_1 \leq 0$
  - in floats: $\varepsilon_1 + 1.0e^{-5} \leq 0$, ie $\varepsilon_1 \leq -1.0e^{-5}$.
- First unstable test possibility :
  - real takes then branch: $\varepsilon_1 \leq 0$
  - float takes else branch: $\varepsilon_1 > -1.0e^{-5}$
  - unstable test = intersection of constraints: $-1.0e^{-5} < \varepsilon_1 \leq 0$
    $$f^y_{[2]} - r^y_{[1]} = (2+\varepsilon_1+1.0e^{-5}) - (4+\varepsilon_1) = -2+1.0e^{-5}.$$
- Second unstable test possibility: conditions $\varepsilon_1 \leq -1.0e^{-5}$ and $\varepsilon_1 > 0$ are non compatible (no unstable test)

Analysis of second example

```
x := [-1,1] + erreur ulp; [1]
y := [-1,1] + erreur ulp; [2]
if (x < y)
   t = y - x;  // [4]
else
   t = x - y;  // [5]
```

First discontinuity error:

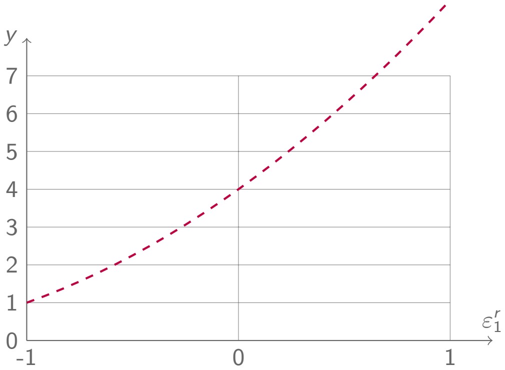
- real takes then branch: $\varepsilon_1^r < \varepsilon_2^r$
- float takes if branch: $\varepsilon_1^r + u\varepsilon_1^e \geq \varepsilon_2^r + u\varepsilon_2^e$ (where $u = ulp(1)$)
- intersection (unstable test), computed in intervals
  - using additional slack variable $\eta_1^t = \varepsilon_1^r - \varepsilon_2^r$, which will appear both in the constraints on real and floats (more generally, if we are interested in the test `exp1 op exp2`, we will associate a slack variable to `exp1-exp2`).
  - we then get $-2u < \varepsilon_1^r - \varepsilon_2^r < 0$ and $\varepsilon_2^e \leq \varepsilon_1^e$

and thus $f_{[5]}^t - r_{[4]}^t = 2(\varepsilon_1^r - \varepsilon_2^r) + u(\varepsilon_1^e - \varepsilon_2^e) + 5u\varepsilon_5^e \in [-7u, 7u]$.

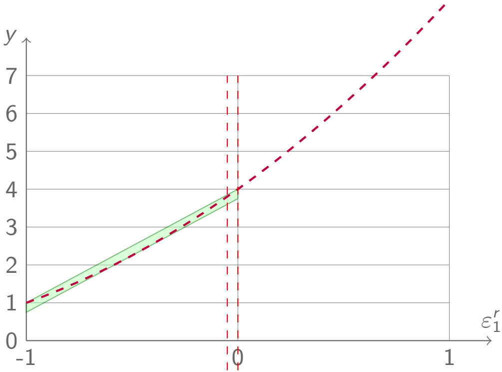
The other discontinuity error is symmetric.

Third example: linearization near the test condition

```
x := [1,3] + 2.5e-6; // [1]
/* f^x_[1] = 2 + ε^r_1 + 2.5e−6 */
if (x ≤ 2)
y = x*x; // [2]
else
y = x*x; // [3]
```



```
x := [1,3] + 2.5e-6; // [1]
/* f_[1]^x = 2 + ε_1^r + 2.5e−6 */
if (x ≤ 2)
y = x*x; // [2]
else
y = x*x; // [3]
```

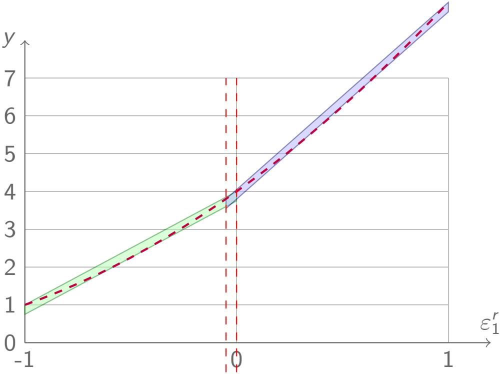


Φ^r: ε_1^r for real value control flow — [2]: $\varepsilon_1^r \leq 0$

Φ^f: ε_1^r for float value control flow — [2]: $\varepsilon_1^r \leq -2.5e^{-6}$

Third example: linearization near the test condition

```
x := [1,3] + 2.5e-6; // [1]
/* f_[1]^x = 2 + ε_1^r + 2.5e-6 */
if (x ≤ 2)
y = x*x; // [2]
else
y = x*x; // [3]
```

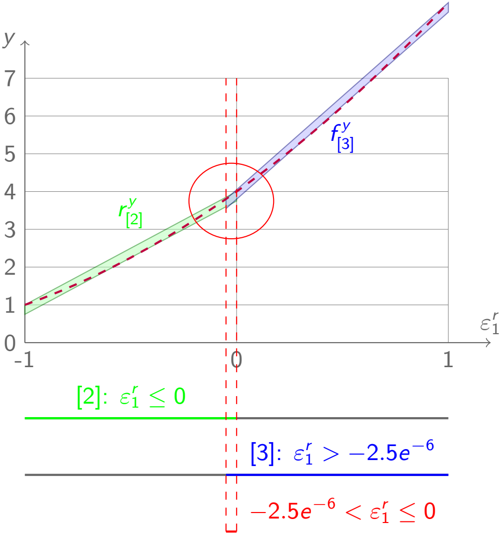


Φ^r: ε_1^r for real value control flow — [2]: $\varepsilon_1^r \leq 0$ — [3]: $\varepsilon_1^r > 0$

Φ^f: ε_1^r for float value control flow — [2]: $\varepsilon_1^r \leq -2.5e^{-6}$ — [3]: $\varepsilon_1^r > -2.5e^{-6}$

Third example: linearization near the test condition

```
x := [1,3] + 2.5e-6; // [1]
/* f[1]^x = 2 + ε1^r + 2.5e^-6 */
if (x ≤ 2)
y = x*x; // [2]
else
y = x*x; // [3]
```



Φ^r: ε_1^r for real value control flow

Φ^f: ε_1^r for float value control flow

$\Phi^r \cap \Phi^f$: ε_1^r for unstable test

Third example

```
x := [1,3] + 2.5e-6;    // [1]
if (x <= 2)
 y = x^2;               // [2]
else
 y = x^2;               // [3]
```

- Unstable test constraint $-2.5e^{-6} < \varepsilon_1^r \leq 0$:
  - the real number takes the `then` branch: $r_{[2]}^x = 2 + \varepsilon_1^r$, $\varepsilon_1^r \leq 0$
  - the float takes the `else` branch $f_{[3]}^x = 2 + \varepsilon_1^r + 2.5e - 6$, $\varepsilon_1^r > -2.5e^{-6}$
  - interpretation of x^2 involves linearization: new noise symbols η_1 and η_2
- Unstable test error

$$f_{[3]}^y - r_{[2]}^y = \delta + e^{-5} + 6.25e^{-12} + (2+5e^{-6})\varepsilon_1^r + (0.125+\delta)\eta_2 - 0.125\eta_1 + u\varepsilon_3^e \quad (3)$$

- No actual discontinuity around the condition, but the non-linearity introduced some loss of accuracy in the analysis: bounds for (3) under constraint $-2.5e^{-6} < \varepsilon_1^r \leq 0$ yield

$$-0.25 + \delta_1 \leq f_{[3]}^x - r_{[2]}^x \leq 0.25 + \delta_2$$

- Need correlation between ε_1^r and η_1 near the boundaries (around $\varepsilon_1^r = 0$)

- Linearization control point [2]:

$$r^y_{[2]} = (1.5+(\varepsilon_1+0.5))^2 = 2.25+3(\varepsilon_1+0.5)+\underbrace{(\varepsilon_1+0.5)^2}_{\in[0,0.25]} = 3.875+3\varepsilon_1+0.125\eta_1$$

with $\eta_1 = 8(\varepsilon_1+0.5)^2 - 1$.

- Use of generalized mean value theorem (like in interval affine forms from SAS'07) around the real value test boundary $\varepsilon_1 = 0$, for $\varepsilon_1 \in [-0.25, 0]$:

$$\hat{\eta}_1(\varepsilon_1) = \eta_1(0) + \Delta\varepsilon_1$$

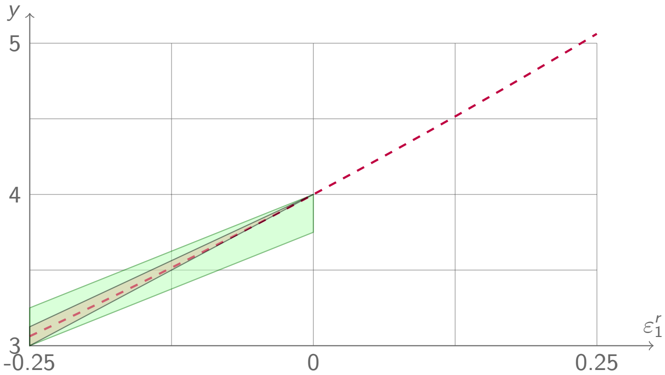
where Δ bounds the derivative $\eta_1'(\varepsilon_1)$ in the range $[-0.25, 0]$. We get

$$\begin{aligned} \hat{\eta}_1 &= 1 + 16([-0.25, 0] + 0.5)\varepsilon_1 \\ &= 1 + [4, 8]\varepsilon_1, \end{aligned}$$

which we can also write

$$1 + 8\varepsilon_1 \leq \eta_1 \leq 1 + 4\varepsilon_1$$

- Same kind of linearization for η_2 introduced in $f^y_{[3]}$ in the else branch



Classical abstraction:

$$r^y_{[2]} = 3.875 + 3\varepsilon_1 + 0.125\eta_1, \ \varepsilon_1 \leq 0$$

Linearization of new symbol near the test condition ($\varepsilon_1 \in [-0.25, 0]$): using $1 + 8\varepsilon_1 \leq \eta_1 \leq 1 + 4\varepsilon_1$ we have

$$4 + 4\varepsilon_1 \leq r^y_{[2]} \leq 4 + 3.5\varepsilon_1 (\varepsilon_1 \in [-0.25, 0])$$

Discontinuity error

Now,

$$f^y_{[3]} - r^y_{[2]} = \delta + e^{-5} + 6.25e^{-12} + (2 + 5e^{-6})\varepsilon^r_1 + (0.125 + \delta)\eta_2 - 0.125\eta_1 + u\varepsilon^e_3$$

with

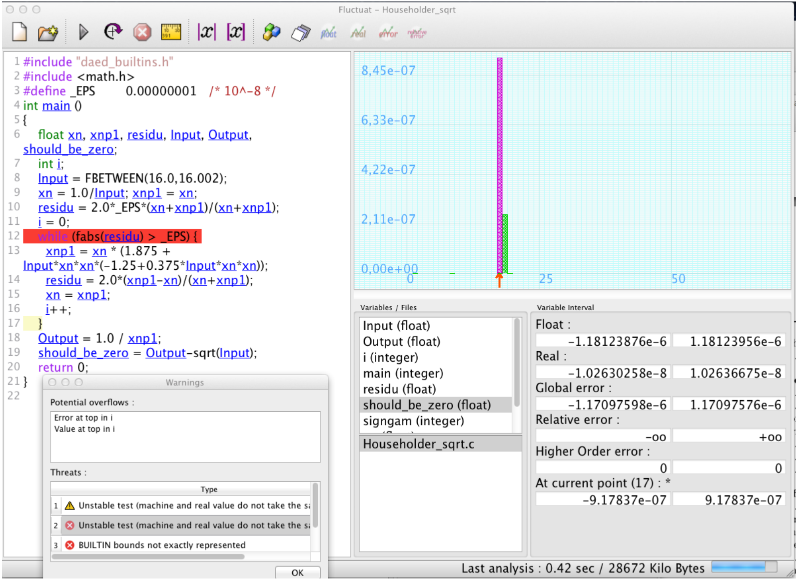
$$\begin{cases} -2.5e^{-6} < \varepsilon^r_1 \leq 0 \\ 1 + 8\varepsilon_1 \leq \eta_1 \leq 1 + 4\varepsilon_1 \\ \frac{0.125-\delta-(1+5e^{-6})\varepsilon_1}{0.125+\delta} \leq \eta_2 \leq \frac{0.125-\delta-0.5\varepsilon_1}{0.125+\delta} \end{cases}$$

gives:

$$f^y_{[3]} - r^y_{[2]} = e^{-5} + 6.25e^{-12} + u\varepsilon^e_3 + [0, (1 + 5e^{-6})\varepsilon^r_1]$$

- Very tight estimate! (actual gap because of the $2.5e^{-6}$ error on the input)
- The decomposition in error in the else branch and discontinuity error can be obtained by the decomposition $f^y_{[3]} = r^y_{[3]} + e^y_{[3]}$.

Householder algorithm for square root



Some useful tools and links for the validation of floating-point computations

- MFPR: multiple precision library with correct rounding (on which Fluctuat relies) http://www.mpfr.org
- CRLIBM: mathematical functions with correct rounding http://lipforge.ens-lyon.fr/projects/crlibm/
- GAPPA, Jessie, Flocq: the Toccata team http://gappa.gforge.inria.fr
  Their gallery of verified fp programs:
  http://toccata.lri.fr/gallery/fp.en.html
- CADNA: stochastic arithmetic to estimate rounding error http://www-pequan.lip6.fr/cadna/
- The workshop on Numerical Software Verification http://www.lix.polytechnique.fr/ ghorbal/NSV-13/

IV. Affine sets: some current and future variations

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, with E. Goubault)
- Noise symbols ε_i being no longer defined in intervals:
  - probabilistic affine forms: ε_i take values in p-boxes (Computing 2012, with O. Bouissou, E. Goubault, J. Goubault-Larrecq)
  - ellipsoids (clear potential for program invariants): $\|\varepsilon\|_2 \leq 1$ (instead of $\|\varepsilon\|_\infty \leq 1$) ?

Motivation for under-approximation

- Existing abstract domains mainly for over-approximation
  - Results are sure but may be pessimistic (“false alarms”)
  - How pessimistic ?
- Under-approximation: sets of values of the outputs, that are sure to be reached for some inputs in the specified ranges
  - Sets of values of the outputs, that are sure to be reached for some inputs in the specified ranges.
- Applications
  - Joint use of under- and over-approximation to characterize the quality of analysis results
  - Extract scenarios giving extreme values

Inner-approximations (Goubault/Putot SAS 2007)

Principle

- Use more general dependency coefficients
  - $\check{x} = \sum_{i=1}^{n} [a_i, b_i]\varepsilon_i$ (modal interval coefficients)
  - Generalized intervals : $\mathbf{x} = [\underline{x}, \overline{x}]$, possibly with $\underline{x} \geq \overline{x}$.

A few words on modal intervals

- $\mathbf{x}$ is proper (in $\mathbb{IR}$) if $\underline{x} \leq \overline{x}$, otherwise improper
- dual $\mathbf{x} = \mathbf{x}^* = [\overline{x}, \underline{x}]$ and pro $\mathbf{x} = [min(\underline{x}, \overline{x}), max(\underline{x}, \overline{x})]$.
- Kaucher arithmetic extending classical interval arithmetic
  - For instance same addition
  - But $[1,2] * [1,-1] = [1,-1]$ whereas $[1,2] *$ pro $[1,-1] = [-2,2]$

Generalized intervals $\mathbf{x} = [\underline{x}, \overline{x}]$, possibly with $\underline{x} \geq \overline{x}$

- dual $\mathbf{x} = [\overline{x}, \underline{x}]$ and pro $\mathbf{x} = [min(\underline{x}, \overline{x}), max(\underline{x}, \overline{x})]$.
- $\mathbf{x}$ is proper (in $\mathbb{IR}$) if $\underline{x} \leq \overline{x}$, otherwise improper
- we note $\boldsymbol{\varepsilon}_\mathbf{i}$ the proper interval including the values of noise symbol ε_i, and $\boldsymbol{\varepsilon}_\mathbf{i}^*$ its dual.

Interpretation with quantifiers, in particular:

- Classical over-approximated interval computation : all intervals are proper

$$(\forall x \in \mathbf{x})\,(\exists z \in \mathbf{z})\,(f(x) = z).$$

- Under-approximated computation : all intervals are improper

$$(\forall z \in \text{pro } \mathbf{z})\,(\exists x \in \text{pro } \mathbf{x})\,(f(x) = z).$$

Kaucher arithmetic on generalized intervals

Extension of classical interval arithmetic.

Addition :

$$\mathbf{x}+\mathbf{y}=[\underline{x}+\underline{y},\overline{x}+\overline{y}] \quad \mathbf{x}-\mathbf{y}=[\underline{x}-\overline{y},\overline{x}-\underline{y}]$$

Multiplication :

$\mathcal{P}=\{[\underline{x},\overline{x}],\ \underline{x}\geq 0 \wedge \overline{x}\geq 0\}$, $-\mathcal{P}=\{[\underline{x},\overline{x}],\ \underline{x}\leq 0 \wedge \overline{x}\leq 0\}$,
$\mathcal{Z}=\{[\underline{x},\overline{x}],\ \underline{x}\leq 0\leq \overline{x}\}$, dual $\mathcal{Z}=\{[\underline{x},\overline{x}],\ \underline{x}\geq 0\geq \overline{x}\}$.

$\mathbf{x}\times y$	$\mathbf{y}\in\mathcal{P}$	$\mathbf{y}\in\mathcal{Z}$	$\mathbf{y}\in-\mathcal{P}$	$y\in$ dual $\mathcal{Z}$
$\mathbf{x}\in\mathcal{P}$	$[\underline{x}\underline{y},\overline{x}\overline{y}]$	$[\overline{x}\underline{y},\overline{x}\overline{y}]$	$[\overline{x}\underline{y},\underline{x}\overline{y}]$	$[\underline{x}\underline{y},\underline{x}\overline{y}]$
$\mathbf{x}\in\mathcal{Z}$	$[\underline{x}\overline{y},\overline{x}\overline{y}]$	$[\min(\underline{x}\overline{y},\overline{x}\underline{y}),$ $\max(\underline{x}\underline{y},\overline{x}\overline{y})]$	$[\overline{x}\underline{y},\underline{x}\underline{y}]$	0
$\mathbf{x}\in-\mathcal{P}$	$[\underline{x}\overline{y},\overline{x}\underline{y}]$	$[\underline{x}\overline{y},\underline{x}\underline{y}]$	$[\overline{x}\overline{y},\underline{x}\underline{y}]$	$[\overline{x}\overline{y},\overline{x}\underline{y}]$
$\mathbf{x}\in$ dual $\mathcal{Z}$	$[\underline{x}\underline{y},\overline{x}\underline{y}]$	0	$[\overline{x}\overline{y},\underline{x}\overline{y}]$	$[\max(\underline{x}\underline{y},\overline{x}\overline{y}),$ $\min(\underline{x}\overline{y},\overline{x}\underline{y})]$

Under-approximation with Kaucher arithmetic

- An under-approximation of $f(x)$,

$$(\forall z \in \text{pro } \mathbf{z})\,(\exists x \in \text{pro } \mathbf{x})\,(f(x) = z).$$

  can be obtained computing $f(\mathbf{x})$ with Kaucher arithmetic if
  - all intervals $(\mathbf{x_1}, \ldots, \mathbf{x_k})$ constituting $\mathbf{x}$ are impropers
  - *every variable x_i appears at most once in expression $f(x)$ (no dependency between sub-expressions)*
  - the result $f(\mathbf{x})$ is an improper interval
- Application scope is limited
  - an under-approximation of $f(x) = x^2 - x$ for $x \in [2, 3]$ cannot be thus computed

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

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 :

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

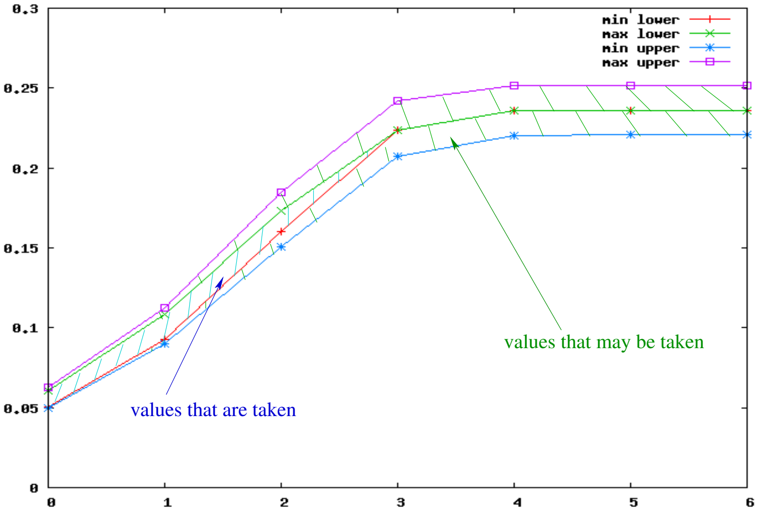
  - Tight enclosure of the iterate :

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

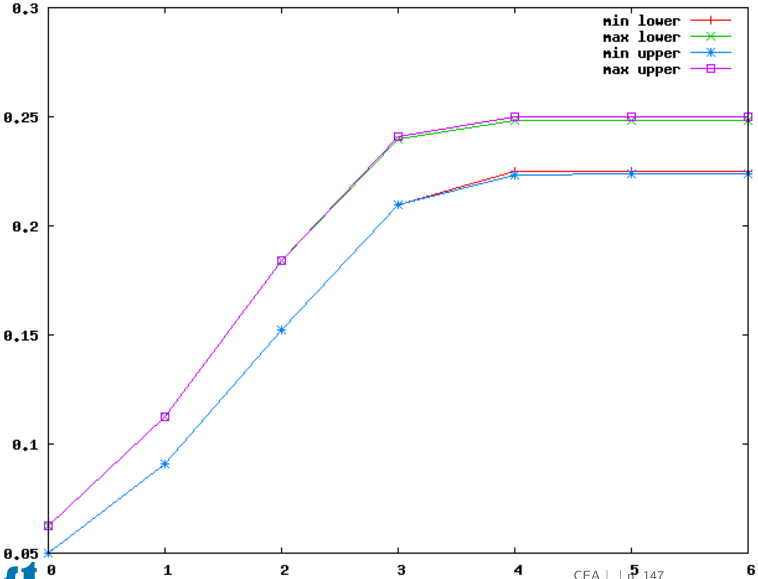
  - Functional proof :

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

Evolution of x_i with iterations (no subdivision)



Evolution of x_i with iterations (8 subdivisions)



- **Linear Filter of order 2**

$$S_i = 0.7E_i - 1.3E_{i-1} + 1.1E_{i-2} + 1.4S_{i-1} - 0.7S_{i-2}$$

where $S_0 = S_1 = 0$, and the E_i are independent inputs in the range $[0, 1]$, that can be modelled by

$$\check{E}_i = \hat{E}_i = \frac{1}{2} + \frac{1}{2}\epsilon_i.$$

- **Fixed unfolding** ($i = 99$) :

$$\begin{aligned} \hat{S}_{99} &= \check{S}_{99} = 0.83 + 7.81e^{-9}\varepsilon_0 - 2.1e^{-8}\varepsilon_1 \\ &\quad + \ldots - 0.16\varepsilon_{98} + 0.35\varepsilon_{99} \end{aligned}$$

  - supposing coefficients computed exactly, gives the exact enclosure of S_{99} : $[-1.0907188500, 2.7573854753]$
  - extreme scenario for S_{99} : the coefficients of the affine form allow us to deduce the E_i leading to the max (or min) of the enclosure ($\alpha_i \geq 0 \Rightarrow E_i = 1$ else $E_i = 0$)

Now: same filter with a non linear perturbation

$$S_i = 0.7E_i - 1.3E_{i-1} + 1.1E_{i-2} + 1.4S_{i-1} - 0.7S_{i-2} + 0.005E_iE_{i-1}$$

- Over-approximation

$$\begin{aligned}\hat{S}_{99} &= 0.837 + 7.81e^{-9}\varepsilon_0 - 2.09e^{-8}\varepsilon_1 + \ldots + 0.351\varepsilon_{99} \\ &\quad +1.77e^{-11}\eta_1 + \ldots + 0.00175\eta_{95} + 0.00125\eta_{96},\end{aligned}$$

terms ε_i correspond to input and are controllable, terms η_j correspond to linearization of non-linear computation and are not

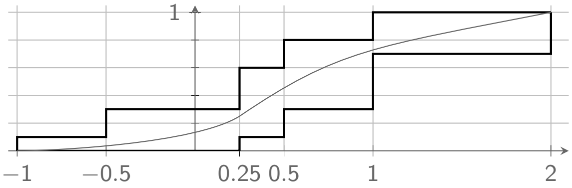
- Under-approximation

$$\begin{aligned}&\check{S}_{99} = 0.837 + 7.81e^{-9}\varepsilon_1 + \ldots + [-0.057, 0.063]\varepsilon_{92} \\ &+[0.07, 0.14]\varepsilon_{93} + [0.18, 0.22]\varepsilon_{94} + [0.25, 0.27]\varepsilon_{95} + [0.22, 0.23]\varepsilon_{96} \\ &+[0.081, 0.087]\varepsilon_{97} + [-0.158, -0.155]\varepsilon_{98} + [0.35, 0.352]\varepsilon_{99}.\end{aligned}$$

- Enclosure $[-0.476, 2.15] \sqsubseteq S_{99} \sqsubseteq [-1.10, 2.77]$
- Extreme scenario : under-approx gives $E_{93} = 1$, $E_{94} = 1$, $E_{95} = 1$, $E_{96} = 1$, $E_{97} = 1$, $E_{98} = 0$, $E_{99} = 1$; and over-approx gives a heuristic for the other inputs, that leads to $S_{99} = 2.766$

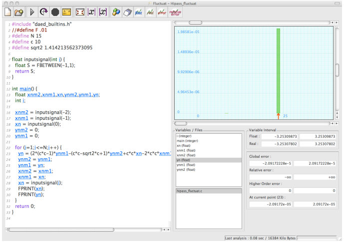
- Some inputs being known set theoretically (non-deterministic inputs) or in probability (probabilistic inputs)
  - e.g. temperature distribution maybe known but we might only know a range for pressure, in some software-driven apparatus
- More generally, inputs may be thought of as given by imprecise probabilities (such as the ones given by probability boxes or P-boxes: pair of upper and lower probabilities)
  - The noise on the input given by some sensor maybe given by a law from statistical physics, depending on a parameter known within an interval. E.g. CCD noise is a gaussian distribution whose variance depends on a temperature, known within an interval.

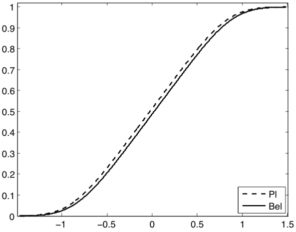
- Discrete p-boxes or Dempster-Shafer structures
  - Generalize probability distributions and interval computations: less pessimistic than intervals but still guaranteed
  - Represent sets of probability distributions: between an upper and a lower Cumulative Distribution Function $P(X \leq x)$



- Encode as much deterministic dependencies as possible by affine arithmetic
  - because arithmetic on p-boxes/DS not very efficient
  - associate a p-box (sets of probability distributions) to each noise symbol instead of [-1,1]
  - both more accurate and faster than direct DS arithmetic

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 (Cumulative Distribution Function, CDF, very close to that of a Gaussian distribution)

References:

- O. Bouissou, E. Goubault, J. Goubault-Larrecq, S. Putot: “A generalization of p-boxes to affine arithmetic” Computing 94 (2012)
- A. Adjé, O. Bouissou, E. Goubault, J. Goubault-Larrecq, S. Putot: “Analyzing Probabilistic Programs with Partially Known Distributions”, VSTTE 2013

- Model "imprecise probabilities"
- Generalize both probabilities and interval computations
  - model for non-deterministic and probabilistic events
- Can be thought of as representations of sets of probability distributions

P-boxes

- Given by upper and lower "probabilities" (CDF form, not necessarily normalized) on $\mathbb{R}$: $\underline{f}$ and $\overline{f}$ from $\mathbb{R}$ to $\mathbb{R}^+$
- for all $x \in \mathbb{R}$, $\underline{f}(x) \leq \overline{f}(x)$

Very similar to an interval domain. Now γ, the concretisation operator is an interval of lower and upper probabilities to a set of probability distributions (instead of a set of values).

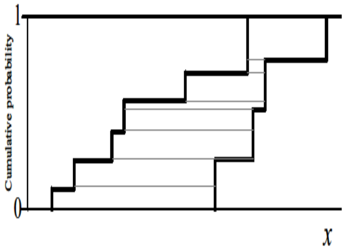
Dempster-Shafer structures

- 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
- 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

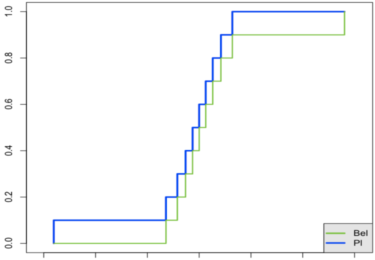
- Determine a belief function Bel and a plausibility function Pl from $\wp(E)$ to $\mathbb{R}$:
  - $Pl(S) = \sum_{T, T \cap S \neq \emptyset} w(T)$
  - $Bel(S) = \sum_{T, T \subseteq S} w(T)$
- $Bel(]-\infty, x]) \leq Pl(]-\infty, x])$ generate a P-box

Given a P-box $(\underline{f}, \overline{f})$

- Subdivide $supp(\underline{f}) \cup supp(\overline{f})$ and take outer approximation by stair functions on this subdivision
- Focal elements and weights can be deduced easily



(taken from SANDIA 2002-4015)



10 subd. on Gaussian(mean=0,std.dev.=0.1)

P-box→DS→P-box gives a “bigger” P-box $(\overline{f}' \geq \overline{f}, \underline{f}' \leq \underline{f})$

Some computation rules: $z = x\Box y$ ($\Box$=+,-,$\times$,/ etc.)

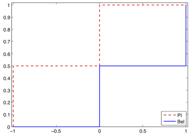
Independent variables x, y (i.e. `xn`, `xnm1` etc.)

- Easy using DS: x (resp. y) given by focal elements F^x (resp. F^y) and weights w^x (resp. w^y)
- Define DS for z: $F^z = \{f^x \Box f^y \mid f^x \in F^X, f^y \in F^y\}$ and $w^z(f^x \Box f^y) = w^x(f^x)w^y(f^y)$ (and renormalize)

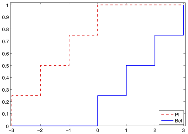
Example

- x with $F^x = \{[-1,0],[0,1]\}$, $w^x([-1,0]) = w^x([0,1]) = \frac{1}{2}$ (approximation of uniform distribution on [-1,1])
- y with $F^y = \{[-2,0],[0,2]\}$, $w^y([-2,0]) = w^y([0,2]) = \frac{1}{2}$

x; y	[-2,0], $\frac{1}{2}$	[0,2], $\frac{1}{2}$
[-1,0], $\frac{1}{2}$	[-3,0], $\frac{1}{4}$	[-1,2], $\frac{1}{4}$
[0,1], $\frac{1}{2}$	[-2,1], $\frac{1}{4}$	[0,3], $\frac{1}{4}$



CDF of x



CDF of $x + y$

Some computation rules (here $\square = +$)

Dependent variables x, y with unknown dependencies (i.e. `yn`, `ynm1` etc.)

- Easy using P-boxes (consequence of Fréchet bounds): x (resp. y) given by upper and lower probabilities $(\overline{f}^x, \underline{f}^x)$ (resp. $(\overline{f}^y, \underline{f}^y)$)
- Define P-box for z:

$$\overline{f}^z(x) = \inf_{u+v=x} \min\left(\overline{f}^x(u) + \overline{f}^y(v), 1\right)$$

$$\underline{f}^z(x) = \sup_{u+v=x} \max\left(\underline{f}^x(u) + \underline{f}^y(v) - 1, 0\right)$$

- Use transfo DS$\leftrightarrow$P-box to find the right formulas on DS directly or use Williamson and Downs/Ferson et al. (LP)/Berleant et al. (what we use currently in our implementation)

Join on DS structures

Given two DSI d_X and d_Y, their join is:

- union of all focal elements from d_X and d_Y, with the same probabilities,
- followed potentially by a normalization if the sum of all probabilities is greater than 1

Example

The join of $d_x = \{\langle[-1,0],0.5\rangle\,;\langle[0,1],0.4\rangle\}$ and $d_y = \{\langle[0.5,1.5],0.2\rangle\}$ is $\{\langle[-1,0],0.46\rangle\,;\langle[0,1],0.36\rangle\,;\langle[0.5,1.5],0.18\rangle\}$.

$\mathtt{lt}_d(d_x, d_y)$ reducing DSI on X for interpreting $X \leq Y$:

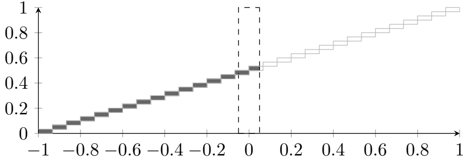
- Resulting DSI contains all the focal elements of the form $\mathtt{lt}_{\mathbb{IR}}(\mathbf{x_i}, \mathbf{y_j})$, when $\langle \mathbf{x_i}, a_i \rangle$ is a focal element of d_x and $\langle \mathbf{y_j}, b_j \rangle$ is a focal element of d_y
- with:

$$\mathtt{lt}_{\mathbb{IR}}([a,b],[c,d]) = \begin{cases} \emptyset & \text{if } a > d \\ [a, \min(b,d)] & \text{otherwise} \end{cases}$$

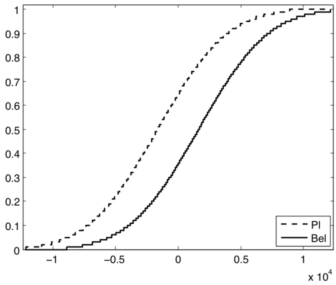
- and the associated probability is then $w_i \times w_j$

Example

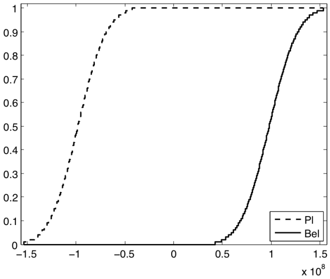
d_1 (gray, below): DSI over-approximating a uniform distribution on $[-1,1]$, d_2 (dotted, below): DSI with one focal element $[-0.05, 0.05]$ (i.e. mimicking a Dirac at 0), then: $\mathtt{lt}_d(d_x, d_y)$ is:



- Suffer from the same disease as interval computations (wrapping effect), and costly computations (here using Matlab/IPPtoolbox by P. Limburg - 100 subd. for each uniform distrib.):



15 iterations (4.65 seconds)



30 iterations (9.30 seconds)

Our approach

- Encode as much deterministic dependencies as possible

```
yn = ...*ynm1-...*ynm2 // not unknown dep.
+...*xn-...*xnm1+...*xnm2); // indep.
```

  - use affine arithmetic based abstraction
  - linearization of dependencies
  - representation on a basis of independent noise symbols
- Use P-boxes for probabilistic values, independent as much as possible...
  - associate a P-box to each noise symbol
  - technicality: some noise symbols (coming from non-linear terms in particular) have unknown dependencies...
- Both more accurate and computationally efficient than pure P-box arithmetic

Affine P-boxes

P-forms

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

Remarks

- Notation: for each program variable x associate

$$\hat{x} = c_0^x + \sum_{i=1}^{n} c_i^x \varepsilon_i + \sum_{j=1}^{m} p_j^x \eta_j$$

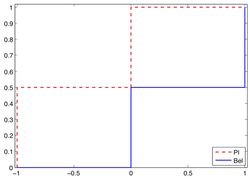
- Experiments in what follows using our C++ implementation

Operator γ

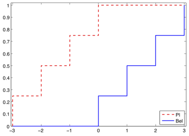
- Easy: use computation rules for addition on independent ε_i and for η_j with unknown dependency (Fréchet bounds)

Example

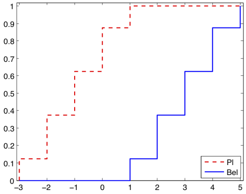
Consider $\hat{x} = 1 + \varepsilon_1 - 2\varepsilon_2 + \eta_1$, all noise symbols being uniform distributions on $[-1, 1]$, approximated by $F = \{[-1, 0], [0, 1]\}$, $w([-1, 0]) = w([0, 1]) = \frac{1}{2}$



CDF of ε_1



CDF of $\varepsilon_1 - 2\varepsilon_2$

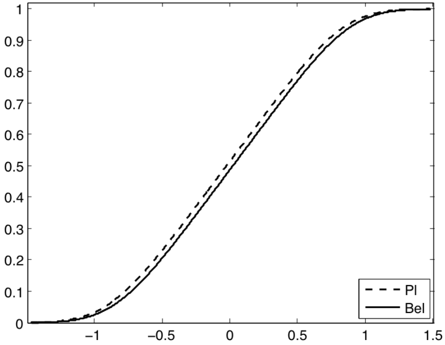


CDF of $\hat{x}$

Linear transformations

- Easy: no new noise symbol, exact linear transformation on the affine (deterministic) part

Example: high-pass filter, 30 iterations - 100 subd. for ε_i



Multiplication

- Linear term (on ε_i) is easy, same as for affine forms
- Associate a new noise symbol η_{m+1} to the non-linear terms:

$$a = \sum_{1 \leq r,l \leq n} c_r^x c_l^y \varepsilon_r \varepsilon_l + \sum_{1 \leq r,l \leq m} p_r^x p_l^y \eta_r \eta_l + \sum_{1 \leq r,l \leq n,m} (c_r^x p_l^y + c_r^y p_l^x) \epsilon_r \eta_l \ .$$

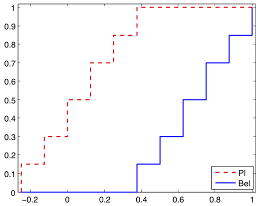
- Associate to η_{m+1} the correct DS, based on the following facts:
  (1) $\epsilon_r \epsilon_l$ ($r \neq l$) is a product of two independent DS
  (2) $\eta_r \eta_l$ ($l \neq r$) and $\epsilon_r \eta_l$ are products of two DS whose dependency is unknown.
  (3) $\epsilon_r \epsilon_l$ with $r = l$ and $\eta_r \eta_l$ with $r = l$ are products of two P-boxes whose dependency is perfectly known (squares).

Iterated power on uniform laws

```
float x in [0,1]; float y in [0,1];
for (int i=1;i<10;i++) x = x*y;
```

Initially - small dependence between x and y:

- $\hat{x} = \frac{1}{2} + \frac{1}{2}\varepsilon_1$,
- $\hat{y} = \frac{1}{2} + \frac{1}{4}\varepsilon_1 + \frac{1}{4}\varepsilon_2$, ε_1 has $F^1 = \{[-1,0],[0,1]\}$, $w^1([-1,0]) = w^1([0,1] = \frac{1}{2}$ (same for ε_2)



Concentrates around 0
Outer approximation but keeps dependencies!

Join on probabilistic affine forms

- Consider:

$$\begin{cases} \hat{x}_1 & = & \alpha_0^1 + \sum_{i=1}^{n} \alpha_i^1 \varepsilon_i + \sum_{j=1}^{m} \beta_j^1 \eta_j \\ \hat{x}_2 & = & \alpha_0^2 + \sum_{i=1}^{n} \alpha_i^2 \varepsilon_i + \sum_{j=1}^{m} \beta_j^2 \eta_j \end{cases}$$

- Join $\hat{x}$ is $\hat{x} = \hat{x}_l + \eta_{m+1}$ with

$$\begin{cases} \hat{x}_l & = & \alpha^0 + \sum_{i=1}^{n} \alpha^i \varepsilon_i + \sum_{j=1}^{m} \beta^j \eta_j \\ \alpha^0 & = & m\big(\gamma_d(\alpha_1^0) \curlyvee \gamma_d(\alpha_1^0)\big) \\ \alpha^i & = & \text{argmin}(\alpha_1^i, \alpha_2^i), \ \forall i \in [1, n] \\ \beta^j & = & \text{argmin}(\beta_1^j, \beta_2^j), \ \forall j \in [1, m] \end{cases}$$

($m(d)$ is the middle of the support of DSI d)

- The new noise symbol η_{m+1} is given by its DSI:

$$d_{\eta_{m+1}} = \gamma_d(\hat{x}_l - x) \curlyvee \gamma_d(\hat{x}_l - y)$$

Interpretation of tests on probabilistic affine forms

- Idea: consider two probabilistic affine forms $\hat{x}$ and $\hat{y}$ over two noise symbols ε_1 and ε_2
- Example: we want to enforce that
  $\alpha_0^x + \alpha_1^x d_{\varepsilon_1} + \alpha_2^x d_{\varepsilon_2} \leq \alpha_0^y + \alpha_1^y d_{\varepsilon_1^y} + \alpha_2^y d_{\varepsilon_2^y}$
- Leads to the following reduction:

$$\begin{aligned} d_{\varepsilon_1} &= \mathtt{lt}_d\left(d_{\varepsilon_1}, \frac{\alpha_0^x - \alpha_0^y + (\alpha_2^x - \alpha_2^y) d_{\varepsilon_2}}{\alpha_1^x - \alpha_1^y}\right) \\ d_{\varepsilon_2} &= \mathtt{lt}_d\left(d_{\varepsilon_2}, \frac{\alpha_0^x - \alpha_0^y + (\alpha_1^x - \alpha_1^y) d_{\varepsilon_1}}{\alpha_2^x - \alpha_2^y}\right) \end{aligned}$$

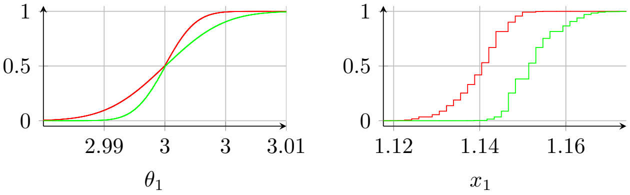
  These equations can be iterated to reduce the DSI associated to ε_1 and ε_2, and we define $\mathtt{lt}(\hat{x}, \hat{y})$ as the greatest fixpoint of the iteration of these two equations.

Experiments: 1 - Ferson polynomial

- 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 $[-0.01, 0.01]$
- Use of VSPODE to obtain a Taylor model polynomial that expresses the solution at $t_f = 20$ as an order 5 polynomial of θ_1 and θ_2
- 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 $P(x_1 \leq 1.13) \leq 0.0552$

Some references

Abstract domains

- Bouissou, Goubault, Goubault-Larrecq and Putot, A generalization of p-boxes to affine arithmetic, Computing 2012
- Goubault and Putot, Static Analysis of Finite Precision Computations, Proceedings of VMCAI'11
- Ghorbal, Goubault and Putot, A Logical Product to Zonotope Intersection, Proceedings of CAV'10
- Ghorbal, Goubault and Putot, The Zonotope Abstract Domain Taylor1+, Proceedings of CAV'09
- Goubault and Putot, A zonotopic framework for functional abstractions, Arxiv 2009
- Goubault and Putot, Perturbed affine arithmetic for invariant computation in numerical program analysis, Arxiv 2008
- Goubault and Putot, Under-Approximations of Computations in Real Numbers Based on Generalized Affine Arithmetic, Proceedings of SAS'07
- Goubault and Putot, Static Analysis of Numerical Algorithms, Proceedings of SAS'06

Around Fluctuat, industrial case studies

- Bouissou, Goubault, Putot, Tekkal and Védrine, HybridFluctuat: A Static Analyzer of Numerical Programs within a Continuous Environment, Proceedings of CAV'09
- Delmas, Goubault, Putot, Souyris, Tekkal and Védrine, Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software, Proceedings of FMICS'09
- Space Software Validation using Abstract Interpretation, Proceedings of the Int. Space System Engineering Conference, DASIA'09
- Goubault, Putot, Baufreton and Gassino, Static Analysis of the Accuracy in Control Systems : Principles and Experiments, Proceedings of FMICS'07