PROPAGATION OF ROUNDING ERRORS BY INTERVAL ARITHMETIC AND AFFINE FORMS

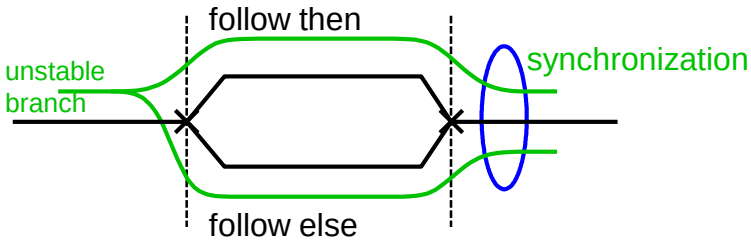
Journée précision numérique | Védrine Franck
Joint work with Eric Goubault & Sylvie Putot & Karim Tekkal + Maxime Jacquemin

DIFFERENT RESULTS OF ACCURACY ANALYSIS

- **Observed accuracy – comparison float ↔ ideal on a test case**
  - **Tools**: **FpDebug**, **MPFR** instrumentation
  - **Properties**: no false alarm, debug capabilities, but underestimate reality
- **Stochastic Intended accuracy – comparison float ↔ float by perturbing every floating-point operation**
  - **Tools**: synchronous stochastic methods – **Cadna**
  - **Tools**: asynchronous monte-carlo methods – **Verificarlo**, **Verrou**
  - **Property**: realistic results, debugging capabilities, but no worst case
- **Guaranteed Intended accuracy – small interval propagation**
  - **Tools**: synchronous abstract interpretation on numerical scenarios – **Fluctuat**, **FLDLib**
  - **Property**: Guaranteed results around a test case, annotations for unstable branches, but overestimates reality
- **Proved accuracy – large interval propagation**
  - **Tools**: provers – **Gappa**
  - **Property**: Guaranteed results, but large number of annotations required

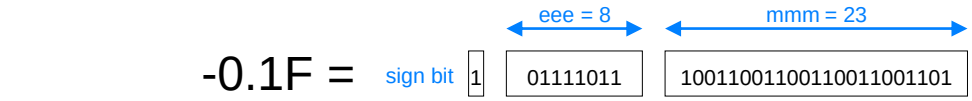
DIFFERENT KINDS OF NUMERICAL ERRORS

- **sudden big loss of accuracy**
  - Cancellation: a ~ b $\Rightarrow$ a - b has few significant digits
- **sudden discontinuity**
  - if ($x \leq -\varepsilon$ or $x \geq \varepsilon$) then $y \leftarrow 1/x$ else $y \leftarrow 0$



- **progressive loss of accuracy**
  - absorption: If *a+x=x* then *a=0* is False!
    - $10^{-20} + 1 = 1$ but $10^{-20} \neq 0$
  - approximated constants

-0.1F =



- **with non-determinism**
  - Associativity: $a + (b + c) \neq (a + b) + c$ for floating-point
    - *(1.10 + 3.30) + 5.50 = 9.900000*
    - *1.10 + (3.30 + 5.50) = 9.900001*
- **model/method error**

ACCURACY ANALYSIS METHODOLOGY

- **From a test case to an analysis with input ranges**
  - First = quantitative results, even if optimistic
  - The investigations can stop: you have some results

- **Verification of the results**
  - Observed accuracy ≤ Stochastic intended ≤ Guaranteed intended ≤ Proved
  - If **Cadna** results < **Verificarlo/Verrou** results then
    - you might investigate unstable branches
    - you might add manual annotations for synchronization

- **Origin of errors**
  - **Herbgrind**: sudden big loss of accuracy
  - **Verrou** with delta-debugging: loss of accuracy at function level
  - **Fluctuat** for small use-cases: progressive loss of accuracy

INTEREST OF GUARANTEED ANALYSES

- **Analysis on a range of inputs**
  - all inputs at the same time
  - risk: associate the results obtained with an input to another input
- **A step before a proof “for all cases”**
- **Show the parts of the code where the developer has written interesting code in interaction with accuracy**
  - ex: a possible division by zero invalidates quantitative accuracy results
  - hard for an automated tool to detect all patterns
  - $\Rightarrow$ enable some annotations to improve the analysis results

INTERVAL & AFFINE FORMS

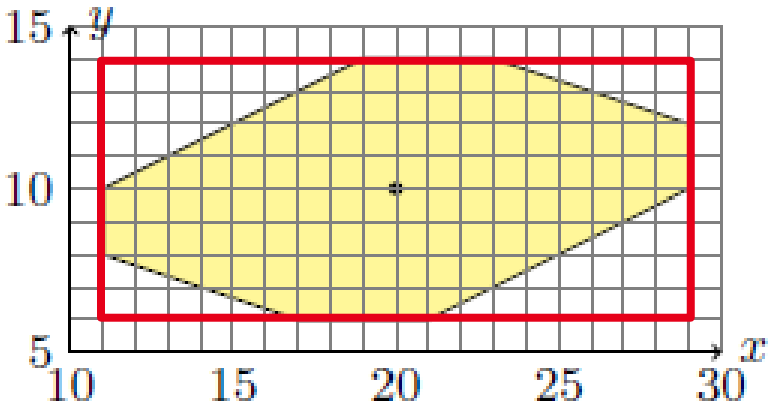
- **A single variable**
  - Interval: $\hat{x} \stackrel{\text{def}}{=} [x_{min}, x_{max}]$ affine form: $\hat{x} \stackrel{\text{def}}{=} \alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i$

    where α_i, x_{min}, x_{max} = constants and ε_i = formal noise symbols $\in$ [-1.0, 1.0]

  - x = [11, 29] $x = 20 + 9\times\varepsilon_1$ or $x = 20 + 2\times\varepsilon_1 + 4\times\varepsilon_3 + 3\times\varepsilon_4$

- **Relationships between several variables**
  - Affine forms store some linear relationships between variables

$x = 20 + 2\times\varepsilon_1 + 4\times\varepsilon_3 + 3\times\varepsilon_4$
$y = 10 + 1\times\varepsilon_2 + 2\times\varepsilon_3 - 1\times\varepsilon_4$



ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

x ← [-2, 4] y ← [-6, 6]

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

$x \leftarrow [-2, 4]$ $y \leftarrow [-6, 6]$

$a \leftarrow [-4, 8]$

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

x ← [-2, 4] y ← [-6, 6]

a ← [-4, 8]

b ← [-2, 2]

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

x ← [-2, 4] y ← [-6, 6]

a ← [-4, 8]

b ← [-2, 2]

z ← [-6, 10]

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

$x \leftarrow [-2, 4]$ $y \leftarrow [-6, 6]$

$a \leftarrow [-4, 8]$

$b \leftarrow [-2, 2]$

$z \leftarrow [-6, 10]$

$r \leftarrow [-16, 18]$

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

$x \leftarrow [-2, 4]$ $\quad y \leftarrow [-6, 6]$

$x \leftarrow x = 1 + 3\times\varepsilon_1$ $\quad y \leftarrow x = 6\times\varepsilon_2$

$a \leftarrow [-4, 8]$

$b \leftarrow [-2, 2]$

$z \leftarrow [-6, 10]$

$r \leftarrow [-16, 18]$

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

$x \leftarrow [-2, 4]$ $\quad x \leftarrow x = 1 + 3\times\varepsilon_1$

$y \leftarrow [-6, 6]$ $\quad y \leftarrow x = 6\times\varepsilon_2$

$a \leftarrow [-4, 8]$ $\quad a \leftarrow 2 + 6\times\varepsilon_1$

$b \leftarrow [-2, 2]$

$z \leftarrow [-6, 10]$

$r \leftarrow [-16, 18]$

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

$x \leftarrow [-2, 4]$ $\quad x \leftarrow x = 1 + 3\times\varepsilon_1$

$y \leftarrow [-6, 6]$ $\quad y \leftarrow x = 6\times\varepsilon_2$

$a \leftarrow [-4, 8]$ $\quad a \leftarrow 2 + 6\times\varepsilon_1$

$b \leftarrow [-2, 2]$ $\quad b \leftarrow 2\times\varepsilon_2$

$z \leftarrow [-6, 10]$

$r \leftarrow [-16, 18]$

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

$x \leftarrow [-2, 4]$ $\quad x \leftarrow x = 1 + 3\times\varepsilon_1$

$y \leftarrow [-6, 6]$ $\quad y \leftarrow x = 6\times\varepsilon_2$

$a \leftarrow [-4, 8]$ $\quad a \leftarrow 2 + 6\times\varepsilon_1$

$b \leftarrow [-2, 2]$ $\quad b \leftarrow 2\times\varepsilon_2$

$z \leftarrow [-6, 10]$ $\quad z \leftarrow 2 + 6\times\varepsilon_1 + 2\times\varepsilon_2$

$r \leftarrow [-16, 18]$

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

$x \leftarrow [-2, 4]$	$y \leftarrow [-6, 6]$
$x \leftarrow x = 1 + 3\times\varepsilon_1$	$y \leftarrow x = 6\times\varepsilon_2$
$a \leftarrow [-4, 8]$	$a \leftarrow 2 + 6\times\varepsilon_1$
$b \leftarrow [-2, 2]$	$b \leftarrow 2\times\varepsilon_2$
$z \leftarrow [-6, 10]$	$z \leftarrow 2 + 6\times\varepsilon_1 + 2\times\varepsilon_2$
$r \leftarrow [-16, 18]$	$r \leftarrow 1 + 3\times\varepsilon_1 - 4\times\varepsilon_2$

- **Results**

ARITHMETIC WITH INTERVALS AND AFFINE FORMS

- **Domain propagation**

```
real f(real x, real y)
{
  real a, b, z, r ;
  a = 2 * x ;
  b = y / 3 ;
  z = a + b ;
  r = z - x - y ;
  return r ;
}
```

	$x \leftarrow [-2, 4]$	$y \leftarrow [-6, 6]$
	$x \leftarrow x = 1 + 3\times\varepsilon_1$	$y \leftarrow x = 6\times\varepsilon_2$
a	$a \leftarrow [-4, 8]$	$a \leftarrow 2 + 6\times\varepsilon_1$
b	$b \leftarrow [-2, 2]$	$b \leftarrow 2\times\varepsilon_2$
z	$z \leftarrow [-6, 10]$	$z \leftarrow 2 + 6\times\varepsilon_1 + 2\times\varepsilon_2$
r	$r \leftarrow [-16, 18]$	$r \leftarrow 1 + 3\times\varepsilon_1 - 4\times\varepsilon_2$

- **Results**

$r \leftarrow [-16, 18]$	$r \leftarrow [-6, 8]$

OVER-APPROXIMATIONS AND SUBDIVISIONS

- **Code**

```
real x ← [0.0, 1.0], y;
```

OVER-APPROXIMATIONS AND SUBDIVISIONS

- **Code**

```
real x ← [0.0, 1.0], y;
/* x_real : 0.5 + 0.5 × ε1 */
```

OVER-APPROXIMATIONS AND SUBDIVISIONS

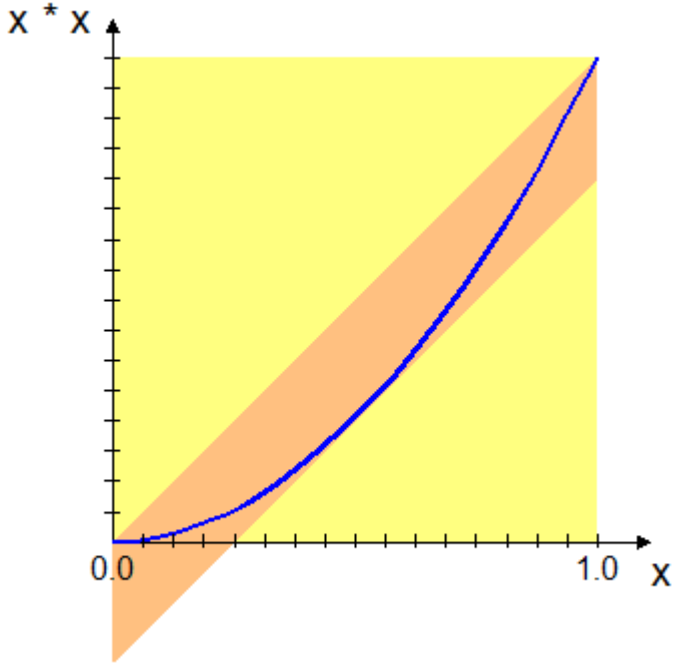
- **Code**

```
real x ← [0.0, 1.0], y;
/* x_real : 0.5 + 0.5 × ε1 */

y = x*x;
/* y_real : 0.375 + 0.5 × ε1 + 0.125 × μ1 intersection with [0.0, 1.0] */
```

OVER-APPROXIMATIONS AND SUBDIVISIONS

- **Code**

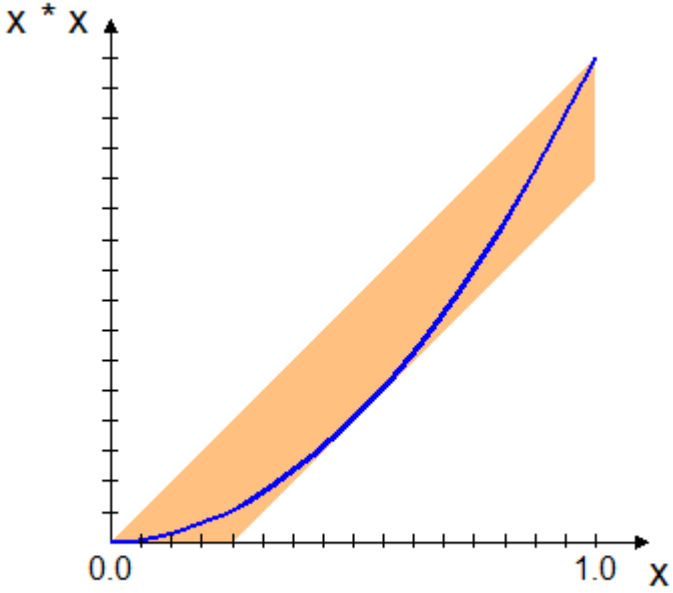
```
real x ← [0.0, 1.0], y;
/* x_real : 0.5 + 0.5 × ε1 */

y = x*x;
/* y_real : 0.375 + 0.5 × ε1 + 0.125 × μ1 intersection with [0.0, 1.0] */
```



OVER-APPROXIMATIONS AND SUBDIVISIONS

- **Code**

```
real x ← [0.0, 1.0], y;
/* x_real : 0.5 + 0.5 × ε1 */

y = x*x;
/* y_real : 0.375 + 0.5 × ε1 + 0.125 × μ1 intersection with [0.0, 1.0] */

y = x-y;
/* y_real : 0.125 - 0.125 × μ1 */
```

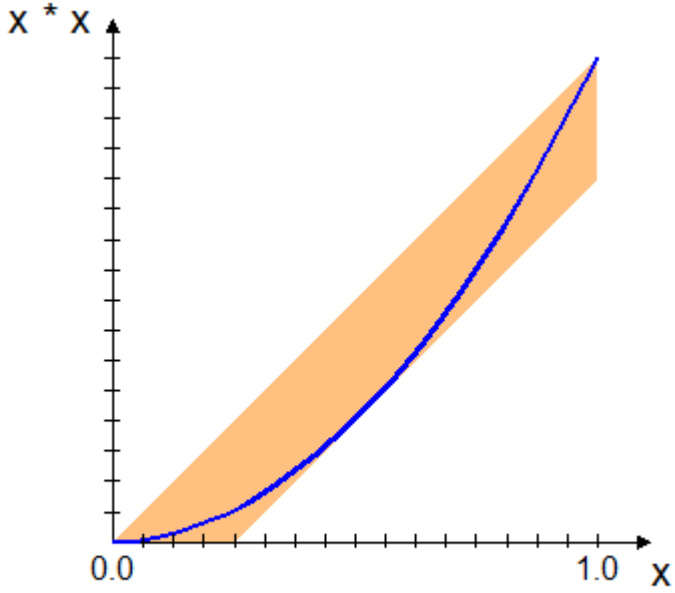


OVER-APPROXIMATIONS AND SUBDIVISIONS

- **Code**

```
real x ← [0.0, 1.0], y;
/* x_real : 0.5 + 0.5 × ε1 */

y = x*x;
/* y_real : 0.375 + 0.5 × ε1 + 0.125 × μ1 intersection with [0.0, 1.0] */

y = x-y;
/* y_real : 0.125 - 0.125 × μ1 */
```

- **with 2 subdivisions**

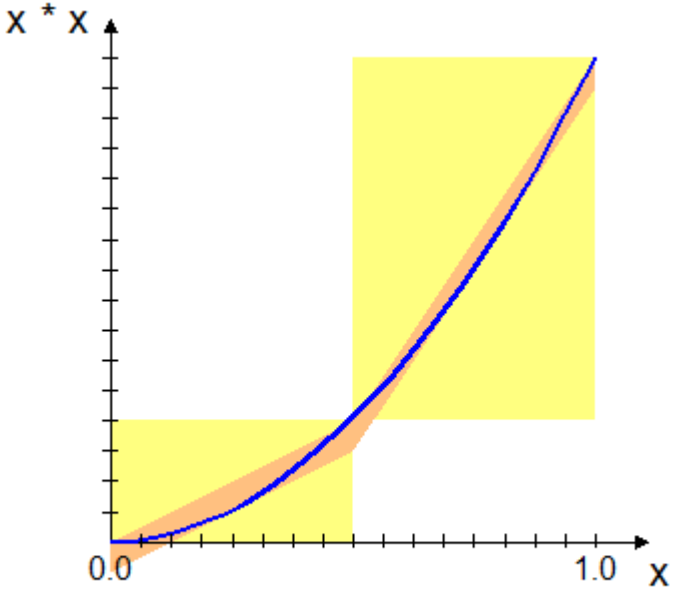


OVER-APPROXIMATIONS AND SUBDIVISIONS

- **Code**

```
real x ← [0.0, 1.0], y;
/* x_real : 0.5 + 0.5 × ε1 */

y = x*x;
/* y_real : 0.375 + 0.5 × ε1 + 0.125 × μ1 intersection with [0.0, 1.0] */

y = x-y;
/* y_real : 0.125 - 0.125 × μ1 */
```

- **with 2 subdivisions**

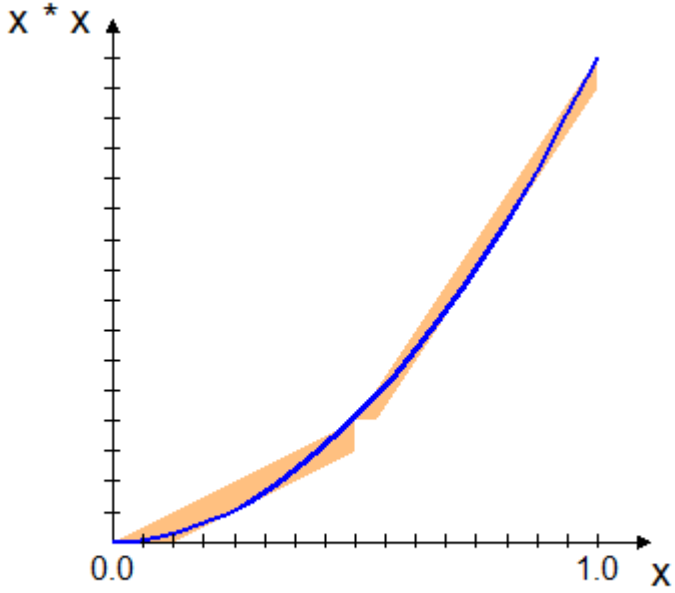


OVER-APPROXIMATIONS AND SUBDIVISIONS

- **Code**

```
real x ← [0.0, 1.0], y;
/* x_real : 0.5 + 0.5 × ε1 */

y = x*x;
/* y_real : 0.375 + 0.5 × ε1 + 0.125 × μ1 intersection with [0.0, 1.0] */

y = x-y;
/* y_real : 0.125 - 0.125 × μ1 */
```

- **with 2 subdivisions**



AFFINE FORMS & NON-LINEAR COMPUTATIONS

- **multiplication**

$$\left(\alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i\right)\left(\alpha_0^y + \sum_{i=1}^{n} \alpha_i^y \varepsilon_i\right) = \alpha_0^x \alpha_0^y + \sum_{i=1}^{n} (\alpha_0^x \alpha_i^y + \alpha_0^y \alpha_i^x)\varepsilon_i + \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i^x \alpha_j^y \varepsilon_i \varepsilon_j$$

AFFINE FORMS & NON-LINEAR COMPUTATIONS

- **multiplication**

$$\left(\alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i\right)\left(\alpha_0^y + \sum_{i=1}^{n} \alpha_i^y \varepsilon_i\right) = \alpha_0^x \alpha_0^y + \sum_{i=1}^{n} (\alpha_0^x \alpha_i^y + \alpha_0^y \alpha_i^x)\varepsilon_i + \boxed{\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i^x \alpha_j^y \varepsilon_i \varepsilon_j}$$

not an affine form $\Rightarrow$ interval approximation $\Rightarrow$ fresh noise symbol

AFFINE FORMS & NON-LINEAR COMPUTATIONS

- **multiplication**

$$\left(\alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i\right)\left(\alpha_0^y + \sum_{i=1}^{n} \alpha_i^y \varepsilon_i\right) = \alpha_0^x \alpha_0^y + \sum_{i=1}^{n} (\alpha_0^x \alpha_i^y + \alpha_0^y \alpha_i^x)\varepsilon_i + \boxed{\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i^x \alpha_j^y \varepsilon_i \varepsilon_j}$$

not an affine form $\Rightarrow$ interval approximation $\Rightarrow$ fresh noise symbol

- **inverse**

AFFINE FORMS & NON-LINEAR COMPUTATIONS

- **multiplication**

$$\left(\alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i\right)\left(\alpha_0^y + \sum_{i=1}^{n} \alpha_i^y \varepsilon_i\right) = \alpha_0^x\alpha_0^y + \sum_{i=1}^{n}(\alpha_0^x\alpha_i^y + \alpha_0^y\alpha_i^x)\varepsilon_i + \boxed{\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i^x\alpha_j^y\varepsilon_i\varepsilon_j}$$

not an affine form $\Rightarrow$ interval approximation $\Rightarrow$ fresh noise symbol

- **inverse**

$$\begin{aligned}
\frac{1}{x} &= \frac{1}{\alpha_0^x} \times \frac{1}{1 + \sum_{i=1}^{n} \frac{\alpha_i^x}{\alpha_0^x}\epsilon_i} \\
&= \frac{1}{\alpha_0^x} \times \frac{1}{1+t} \quad \texttt{with } t = \sum_{i=1}^{n} \frac{\alpha_i^x}{\alpha_0^x}\epsilon_i \\
&= \frac{1}{\alpha_0^x} \times \left(1 - t + \frac{t^2}{1+t}\right) \\
&= \frac{1}{\alpha_0^x} \times \left(1 - t + [0, \frac{(-t_{\texttt{min}})^2}{1+t_{\texttt{min}}}]\right)
\end{aligned}$$

AFFINE FORMS & NON-LINEAR COMPUTATIONS

- **multiplication**

$$\left(\alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i\right)\left(\alpha_0^y + \sum_{i=1}^{n} \alpha_i^y \varepsilon_i\right) = \alpha_0^x\alpha_0^y + \sum_{i=1}^{n}(\alpha_0^x\alpha_i^y + \alpha_0^y\alpha_i^x)\varepsilon_i + \boxed{\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i^x\alpha_j^y\varepsilon_i\varepsilon_j}$$

not an affine form $\Rightarrow$ interval approximation $\Rightarrow$ fresh noise symbol

- **inverse**

$$\begin{aligned}
\frac{1}{x} &= \frac{1}{\alpha_0^x} \times \frac{1}{1 + \sum_{i=1}^{n} \frac{\alpha_i^x}{\alpha_0^x}\epsilon_i} \\
&= \frac{1}{\alpha_0^x} \times \frac{1}{1+t} \quad \texttt{with } t = \sum_{i=1}^{n} \frac{\alpha_i^x}{\alpha_0^x}\epsilon_i \\
&= \frac{1}{\alpha_0^x} \times \left(1 - t + \frac{t^2}{1+t}\right) \\
&= \frac{1}{\alpha_0^x} \times \left(1 - t + \boxed{[0, \frac{(-t_{\texttt{min}})^2}{1+t_{\texttt{min}}}]}\right)
\end{aligned}$$

ACCURACY ANALYSIS

- **(abstract) interpretation of the code with interval & affine forms**
  - At every program point *p* between two instructions

```
float f(float x, float y)
{
  float a, b, z, r ;
1 a = 2 * x ;
2 b = y / 3 ;
3 z = a + b ;
4 r = z - x - y ;
5 return r ;
}
```

ACCURACY ANALYSIS

- **(abstract) interpretation of the code with interval & affine forms**
  - At every program point *p* between two instructions

    variable → ideal domain
    floating-point domain
    absolute error
    relative error

  - domain, error: affine form ∩ interval

```
float f(float x, float y)
{
  float a, b, z, r ;
1 a = 2 * x ;
2 b = y / 3 ;
3 z = a + b ;
4 r = z - x - y ;
5 return r ;
}
```

ACCURACY ANALYSIS

- **(abstract) interpretation of the code with interval & affine forms**
  - At every program point *p* between two instructions

    variable → ideal domain
    floating-point domain
    absolute error
    relative error

  - domain, error: affine form ∩ interval
  - constraints:
    - absolute error = floating-point - ideal
    - relative error = (absolute error)/ideal

```
float f(float x, float y)
{
  float a, b, z, r ;
1 a = 2 * x ;
2 b = y / 3 ;
3 z = a + b ;
4 r = z - x - y ;
5 return r ;
}
```

ACCURACY ANALYSIS

- **(abstract) interpretation of the code with interval & affine forms**
  - At every program point *p* between two instructions

    variable → ideal domain
    floating-point domain
    absolute error
    relative error

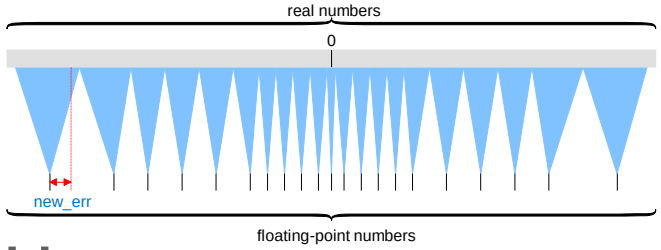
  - domain, error: affine form ∩ interval
  - constraints:
    - absolute error = floating-point - ideal
    - relative error = (absolute error)/ideal
- **Propagation rules for domains and errors**

```
float f(float x, float y)
{
  float a, b, z, r ;
1 a = 2 * x ;
2 b = y / 3 ;
3 z = a + b ;
4 r = z - x - y ;
5 return r ;
}
```

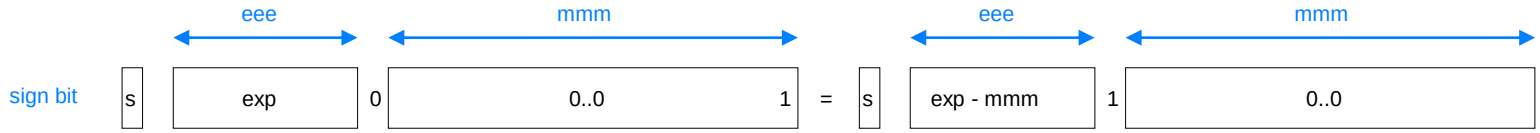
PROPAGATION RULES

- **derivative functions**
  - f(x,y) = f(ideal + err, ideal' + err') = f(ideal, ideal')
    + err $\times$ δf/δy(ideal, ideal') + err' $\times$ δf/δx(ideal, ideal') + non-linear interval
    + new_err due to IEEE-754 operation
- **addition**
  - ideal + err $\oplus$ ideal' + err' = ideal + ideal' + err + err' + new_err
- **multiplication**
  - ideal + err $\otimes$ ideal' + err' = ideal $\times$ ideal' + err $\times$ ideal' + err' $\times$ ideal
    + err $\times$ err' + new_err
- **division**
  - ideal + err $\oslash$ ideal' + err' = ideal / ideal' + err / ideal' - err' $\times$ (ideal / ideal'^{2})
    + non-linear interval + new_err

NEW ERROR INTRODUCTION



- **For every IEEE-754 op = +, -, ×, /, sqrt, fmadd**
  mode = round to nearest (even)
  - new_err $\in$ ulp(0.5) $\times$ [- |floating-point domain|, +|floating-point domain|]
  - for a floating-point val: ulp(val) = max(succ(|val|) - |val|, |val| - pred(|val|))



- **Intended accuracy = accumulation of semi-ulp errors**
  - statistical accumulation for stochastic arithmetic
  - worst-case accumulation for conservative (guaranteed) arithmetic
- **But more reasoning should be used for finer proofs**
  - constant values = compiler constants, error = float - ideal is constant
  - Sterbenz lemma: new_err = 0 for a-b if $0 \leq a/2 \leq b \leq 2 \times a$
  - multiplication, division by 2^n : new_err = 0
  - $a - b \neq 0 \Rightarrow |a-b| \geq 0.5 \times \text{ulp}(\max(|a|, |b|))$

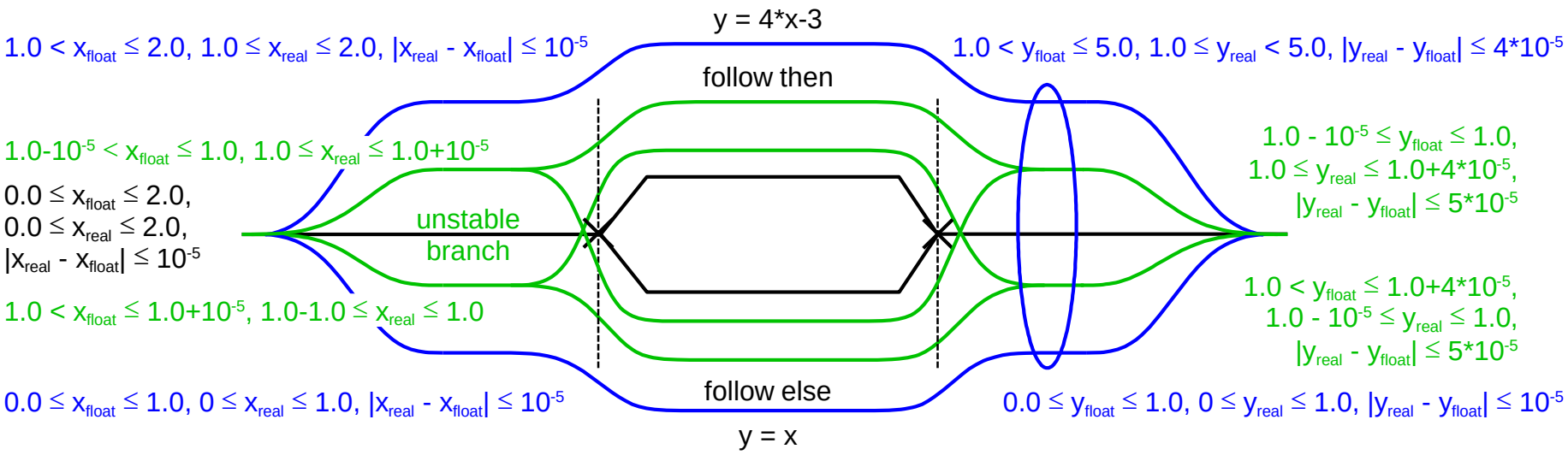
MECHANISMS INSIDE FLUCTUAT

- **Propagation of domain and error for floating-point values (and integer values) for floating-point operations**
- **Constraints to manage conditionals and unstable branches**
- **Widening, narrowing to manage loops**
  - come with parameters: initial unfolding, cyclic unfolding, widening threshold
- **Evolution graphs**
  - help to have feedback on the parameters
- **Interactive analysis**
  - with commands inspired from gdb
  - to understand the warnings of the analysis

CONDITIONS – UNSTABLE BRANCHES

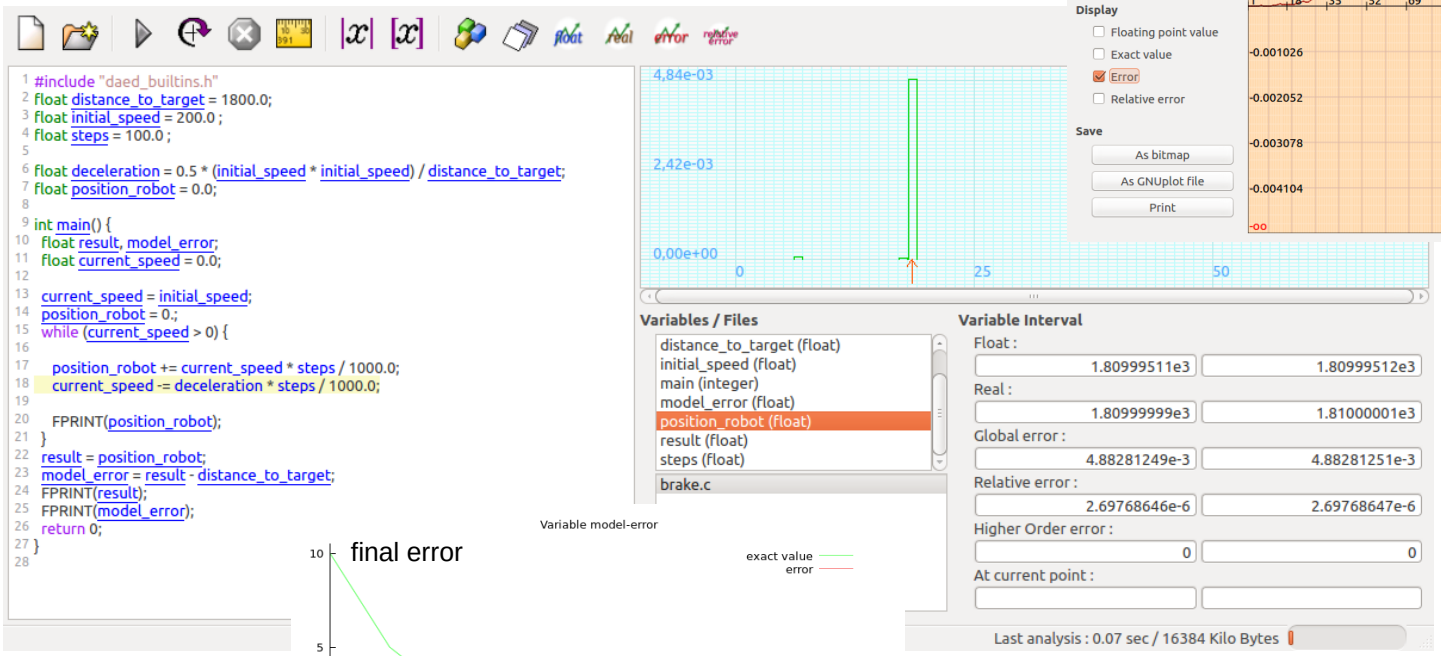
```
x ∈ [0.0, 2.0], error ∈ [-10^-5, 10^-5]
y = (x < 1) ? 4*x-3 : x;
```

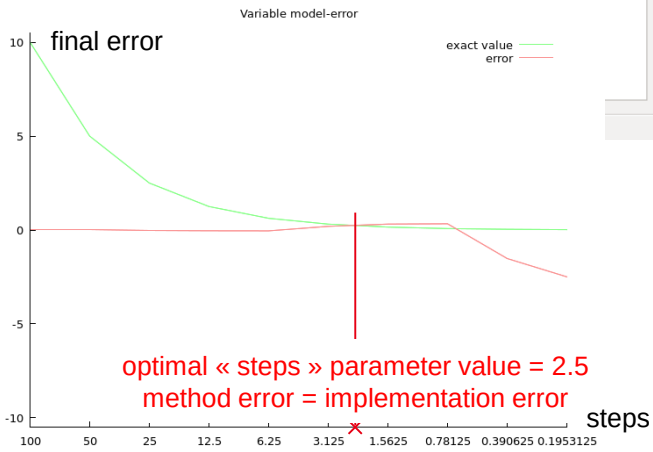
- **constraints the domains for the explored branch**
- **add constraints on noise symbols**
  - example for unstable branch: float < ideal, error < 0
- **synchronization on the immediate post-dominator of the condition**



EVOLUTION OF THE ERROR

- Evolution of the error to tune the analysis parameters





optimal « steps » parameter value = 2.5
method error = implementation error

IRREGULAR SUBDIVISIONS WITH USER'S REQUIREMENTS

- **Specific annotations in the code**

```
double res;
double eps = 1e-8;
double a, b;
double* tab[2] = { &a, &b };
a = DBETWEEN(1, 4);
b = DBETWEEN(1, 4);

SUBDIV_REQUIRE_ERROR_MULTIPLE_BEGIN(tab, 2, res, -1e-5, 1e-5)
DPRINT(a);
DPRINT(b);
if (a < b-eps) {
   res = a*(b/a-1)/log(b/a);
}
else if (a > b+eps) {
   res = a*(b/a-1)/log(b/a);
}
else
  res = a;
DPRINT(res);
SUBDIV_END
```

- **Split the domain to explore 2^n sub-paths where n is the number of variables to subdivide**
- **Do it recursively while the result does not fit with the requirements**

CONCLUSION

- **Analysis is relatively quick – compared to other formal tools**
  - Forward propagation of domain and error by using propagation rules
  - instruction = constant time if the number of noise symbols is limited
    - = linear (+,-,unary), quadratic (*,/) if no constraints on the number of noise symbols

- **Good results for small pieces of code**

- **But guaranteed results mean over-approximations**
  - they accumulate and trigger alarms
  - some of them are true alarms
    - but, do you want to care about events with few probability?
  - some of them are false alarms
    - non-linear computations $\Rightarrow$ the inferred domains are larger than real ones
      $\Rightarrow$ the introduced error (ulp) is over-approximated

    branching are source of false alarms: highly non-linear
    it would require a solver (longer analyses)

Thanks for
your attention!

Commissariat à l'énergie atomique et aux énergies alternatives
Institut List | CEA SACLAY NANO-INNOV | BAT. 861 – PC142
91191 Gif-sur-Yvette Cedex - FRANCE
www-list.cea.fr

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019