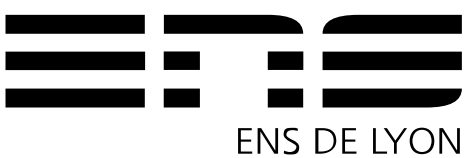
Verifying Code That Uses Error-Free Transformations

Tom Hubrecht, Guillaume Melquiond

June 29th, 2026 – ARITH 2026



Sum all elements of a vector

Naïve method

```c
double s = 0.0;
for (size_t i = 0; i <= n; i++) {
  s = s + x[i];
}
```

$\Rightarrow$ Potential error of $\frac{nu}{1-nu} \sum_{i=0}^{n} |x_i| \approx nu \sum_{i=0}^{n} |x_i| \qquad u := 2^{-p}$

Sum all elements of a vector

Kahan's algorithm (1965)

```c
double s = x[0], c = 0.0;
for (size_t i = 1; i < n; i++) {
  double y = x[i] - c;
  double t = s + y;
  c = (t - s) - y; s = t;
}
```

$\Rightarrow$ Potential error of $\left(2u + \mathcal{O}\left(nu^2\right)\right) \sum_{i=0}^{n} |x_i|$ instead of $nu \sum_{i=0}^{n} |x_i|$

- Much more precise
- $t = s + y$ and $c = (t - s) - y \rightarrow$ what is going on ?

Floating-Point rounding

Rounding Functions

- $\mathcal{F}_p$: Floating-point numbers of precision p
- Several modes defined: to-nearest (with tie breaking), towards ∞, towards 0

$$\circ : \mathbb{R} \to \mathcal{F}_p$$

Recovering the rounding error

- Round-to-nearest, (assuming no overflow/underflow)
  $\Rightarrow (a + b) - \circ(a + b), \quad (a \times b) - \circ(a \times b) \in \mathcal{F}_p$

- Simple algorithms allow computing this error
  - $+ \to$ `TwoSum`, `FastTwoSum`
  - $\times \to$ `TwoProd`

TwoProd

```c
hi = a * b;
lo = fma(a, b, -hi); // a × b - hi
```

→ assuming no underflow, hi + lo = $a \times b$

FastTwoSum

```c
hi = a + b;
t = hi - a;
lo = b - t;
```

→ assuming $|a| > |b|$, hi + lo = $a + b$

How to formally prove algorithms that use those properties ?

- Avoid tedious handwritten proofs
- Use Gappa[1], and extend it

[1] https://gappa.gitlabpages.inria.fr

How to formally prove algorithms that use those properties ?

- Avoid tedious handwritten proofs
- Use Gappa[1], and extend it

A tool to prove properties on numerical programs

- Set of property classes (instanciated in predicates) for real expressions

$$\texttt{BND(x, I)} \iff x \in I$$
$$\texttt{REL(x, y, I)} \iff \exists \varepsilon \in I : x = (1+\varepsilon) \cdot y$$
$$\texttt{FIX(x, k)} \iff \exists m \in \mathbb{Z} : x = m \cdot 2^k$$
$$\texttt{FLT(x, k)} \iff \exists e \in \mathbb{Z}, |m| < 2^k : x = m \cdot 2^e$$

- Set of theorems (~ 300) based on those predicates

$$\texttt{BND(x, I)} \land \texttt{BND(y, J)} \Rightarrow \texttt{BND(x + y, I + J)}$$
$$\texttt{FIX(x, k)} \land \texttt{FIX(y, k')} \Rightarrow \texttt{FIX(x * y, k + k')}$$

[1] https://gappa.gitlabpages.inria.fr

A small example: floating-point squaring

$$z_h + z_\ell \approx (x_h + x_\ell)^2$$

```c
double zh = xh * xh;
double  v = fma(xh, xh, -zh);
double zl = fma(xh * 2, xl, v);                    // xh * 2 is exact
```

Gappa script:

```
@rnd = float<53,ne>;                  # Floating-point arithmetic, unbounded

xh = rnd(x); xl = -(xh - x);                      # Define a double-double
zh = rnd(xh * xh); v = -(zh - xh * xh);                    # Model an EFT
zl = rnd(v + 2 * xh * xl);
```

A small example: floating-point squaring

Prove that the relative error is smaller than $3u^2$ (theoretical bound over $\mathcal{F}_p$)

```
1  { x in [1,2] -> zh + zl -/ x * x in [-3b-106,3b-106] }
2
3  zh + zl - x * x -> - (xl * xl) + (zl - (v + 2 * xh * xl));
```

- $3u^2$ is the optimal general bound
- Errors are explicit
- Input domain bounded 😕

New LIN predicate

$$\texttt{LIN(a, b, I)} \iff \exists \lambda \in I : a = \lambda \cdot b \quad \not\leftrightarrow \quad \frac{a}{b} \in I$$

New LIN predicate

$$\texttt{LIN(a, b, I)} \iff \exists \lambda \in I : a = \lambda \cdot b \quad \rightsquigarrow \quad \frac{a}{b} \in I$$

New theorems

- Addition: $\frac{a}{c} \in I \land \frac{b}{c} \in J \Rightarrow \frac{a+b}{c} \in I + J$
- Product: $\frac{a}{c} \in I \land \frac{b}{d} \in J \Rightarrow \frac{a \times b}{c \times d} \in I * J$
- Generalized relative error: $\frac{a}{c} \in I \land \frac{b-a}{a} \in J \Rightarrow \frac{b-a}{c} \in I * J$

New LIN predicate

$$\texttt{LIN(a, b, I)} \iff \exists \lambda \in I : a = \lambda \cdot b \quad \not\Leftrightarrow \quad \frac{a}{b} \in I$$

New theorems

- Addition: $\frac{a}{c} \in I \land \frac{b}{c} \in J \Rightarrow \frac{a+b}{c} \in I + J$
- Product: $\frac{a}{c} \in I \land \frac{b}{d} \in J \Rightarrow \frac{a \times b}{c \times d} \in I * J$
- Generalized relative error: $\frac{a}{c} \in I \land \frac{b-a}{a} \in J \Rightarrow \frac{b-a}{c} \in I * J$

Reformulate old theorems (e.g. Sterbenz lemma)

$$\frac{a}{b} \in \left[\frac{1}{2}, 2\right] \land \texttt{FLT(a, p)} \land \texttt{FLT(b, p')} \Rightarrow \texttt{FLT(a - b, max(p, p'))}$$

- Represent double-double values: $x := x_h + x_\ell, \quad |x_\ell| \leq 2^{-p} \cdot \mathrm{ufp}(x_h)$

Using the magnitude for error analysis

- Previously in Gappa: $|\circ(x) - x| \in I$
  → bounds the absolute error, `REL` for relative error

- Better rounding errors: $\frac{a}{b} \in I \Rightarrow \frac{\circ(a)-a}{b} \in \left[-2^{-p}, 2^{-p}\right] * I$
- If $b = 2^k, \circ\left(\frac{a}{b}\right) = \frac{\circ(a)}{b}$, and $\frac{a}{b} \in [i_\ell, i_h] \Rightarrow \frac{\circ(a)}{b} \in [\circ(i_\ell), \circ(i_h)]$

Revisiting the squaring routine

Prove that the relative error is smaller than $3u^2$ on the whole domain

```
1  { x <> 0 -> zh + zl -/ x * x in [-3b-106,3b-106] }
```

New hints:

```
1  (zh + v - x*x) / (x*x) -> (zh + v - x*x) / (ufp(x)*ufp(x))
2    / ((x / ufp(x)) * (x / ufp(x))) { x <> 0, ufp(x) <> 0 };
3
4  $ (x / ufp(x)) * (x / ufp(x));
5
6  x*x / (ufp(x)*ufp(x)) -> (x/ufp(x))*(x/ufp(x)) { ufp(x) <> 0 };
```

Recover the overlap: `zl // zh in ?` $\rightarrow \left|\frac{z_\ell}{z_h}\right| < (3+\varepsilon)\cdot u$

Triple-word multiplication

$$u_h + u_l + z_0 + b + c \approx (x_h + x_m + x_\ell) \times (y_h + y_m + y_\ell)$$

```c
double uh = xh * yh, ul = fma(xh, yh, -uh);
double v0h = xh * ym, v0l = fma(xh, ym, -v0h);
double v1h = xm * yh, v1l = fma(xm, yh, -v1h);
double w0 = fma(xh, yl, v0l), w1 = fma(xl, yh, v1l);
double z0 = w0 + w1;
double b = v0h + v1h, b2 = add3(v0h, v1h, -b);
double c = fma(xm, ym, b2);
```

Hand-crafted[1] relative error bound : $28u^3 + 107u^4$

[1] N. Fabiano, J.-M. Muller, and J. Picot, "Algorithms for Triple-Word Arithmetic," *IEEE TC*, vol. 68, no. 11, pp. 1573–1583, Nov. 2019, doi: 10.1109/TC.2019.2918451.

Gappa: Error bound

```
1    |xm // ufp(xh)| <= 1b-53 /\ |xl // ufp(xh)| <= 1b-106 /\
2    |ym // ufp(yh)| <= 1b-53 /\ |yl // ufp(yh)| <= 1b-106 /\
3    xh <> 0 /\ yh <> 0 -> uhl + b + c + z0 -/ x * y in ?
```

- Proves $28u^3 + \mathcal{O}(u^4)$, but without the final normalization ($\sim 2u^3 + \mathcal{O}(u^4)$)
  - Almost as good as the previously published pen-and-paper error bounds
  - Formally checked by the Rocq proof assistant

Gappa: Error bound

```
|xm // ufp(xh)| <= 1b-53 /\ |xl // ufp(xh)| <= 1b-106 /\
|ym // ufp(yh)| <= 1b-53 /\ |yl // ufp(yh)| <= 1b-106 /\
xh <> 0 /\ yh <> 0 -> uhl + b + c + z0 -/ x * y in ?
```

- Proves $28u^3 + \mathcal{O}(u^4)$, but without the final normalization ($\sim 2u^3 + \mathcal{O}(u^4)$)
  - Almost as good as the previously published pen-and-paper error bounds
  - Formally checked by the Rocq proof assistant

Using the same kind of hints as for the square:

```
(uh + ul + b + c + z0) - x * y ->
  - (xm * yl) - (xl * ym) - (xl * yl)
  + (c - (b2 + xm * ym))
  + (z0 - ((v0l + xh * yl) + (v1l + xl * yh)));
```

Experimenting with variants


```c
// double b, b2 = twosum(v0h, v1h, &b);
// double c = fma(xm, ym, b2);
double z1 = fma(xm, ym, z0);
```

⇒ Slightly simpler algorithm, error of $32u^3 + \varepsilon$

Final words

- Extension of the Gappa language:
  - Create easier proofs for algorithms on multi-word arithmetic
- Shown how to help Gappa understand Error-Free Transformations
  - Allows for easier experimentations for different implementations
  - Define overlap on the inputs

In the future:

- Rework classical algorithms to improve known bounds in a simpler manner
  - double-double division: $15u^2$ previously known, for now Gappa gives $11u^2$
- Find out if Gappa can be updated to tackle cancellations in generic algorithms
  - Easier when inputs are constrained, e.g. the coefficient are known in a polynomial evaluation