# Slimmer Formal Proofs for Mathematical Libraries

Paul Geneau de Lamarlière[1, 2], Guillaume Melquiond[2], Florian Faissole[1]

ARITH 2023, September 4th

[1] MITSUBISHI ELECTRIC R&D CENTRE EUROPE
[2] UNIVERSITE PARIS-SACLAY, CNRS, ENS PARIS-SACLAY, INRIA, LMF

MFR2023-ARC-0439

# Mathematical Libraries and Formal Proofs

Much work is done on finding floating-point approximations of real functions:

- Libraries of functions for single (32-bit) and double (64-bit) precision: CORE-MATH, CRLibm, etc.
- Automated generation of such functions: MetaLibm

→ We want these functions to be accurate enough.

Proof with pen and paper is too long and subject to error: need for formal proofs.
But formal proof by hand is still cumbersome.

# Mathematical Libraries and Formal Proofs

Much work is done on finding floating-point approximations of real functions:

- Libraries of functions for single (32-bit) and double (64-bit) precision: CORE-MATH, CRLibm, etc.
- Automated generation of such functions: MetaLibm

$\rightarrow$ We want these functions to be accurate enough.

Proof with pen and paper is too long and subject to error: need for formal proofs.
But formal proof by hand is still cumbersome.

# Correctness of floating-point algorithms

```
def cw_exp(x):
    if x < −746:
        return 0
    else if 710 < x:
        return +∞
    else:
        k  ← nearbyint(x × C)
        t  ← x − k × c1 − k × c2
        t2 ← t × t
        p  ← p0 + t2 × (p1 + t2 × p2)
        q  ← q0 + t2 × (q1 + t2 × q2)
        f  ← (t × p)/(q − t × p) + 1/2
        return f × 2^(k+1)
```

Reduction ($k$, $t$): $C \simeq 1/\ln 2$ and $c_1 + c_2 \simeq \ln 2$

Approximation ($t_2$, $p$, $q$, $f$)

→ Crucial details of correctness proof:

- $t$ is close enough to $x - k\ln 2$: $x - k \times c_1$ is performed exactly
- No exceptional behaviour occurs in the approximation part
- The latter is a good enough approximation

```
def cw_exp(x):
    if x < −746:
        return 0
    else if 710 < x:
        return +∞
    else:
        k  ← nearbyint(x × C)
        t  ← x − k × c1 − k × c2
        t2 ← t × t
        p  ← p0 + t2 × (p1 + t2 × p2)
        q  ← q0 + t2 × (q1 + t2 × q2)
        f  ← (t × p)/(q − t × p) + 1/2
        return f × 2^(k+1)
```

- Lines $k$ and $t$ — Reduction: $C \simeq 1/\ln 2$ and $c_1 + c_2 \simeq \ln 2$
- Lines $t_2$ to $f$ — Approximation

$\rightarrow$ Crucial details of correctness proof:

- $t$ is close enough to $x - k\ln 2$: $x - k \times c_1$ is performed exactly
- No exceptional behaviour occurs in the approximation part
- The latter is a good enough approximation

# Correctness of floating-point algorithms

MITSUBISHI ELECTRIC
*Changes for the Better*

**def** cw_exp($x$):
  **if** $x < -746$:
    **return** $0$
  **else if** $710 < x$:
    **return** $+\infty$
  **else**:

$$
\left.\begin{array}{lcl}
k & \leftarrow & \text{nearbyint}(x \times C) \\
t & \leftarrow & x - k \times c_1 - k \times c_2
\end{array}\right\} \text{Reduction: } C \simeq 1/\ln 2 \text{ and } c_1 + c_2 \simeq \ln 2
$$

$$
\left.\begin{array}{lcl}
t_2 & \leftarrow & t \times t \\
p & \leftarrow & p_0 + t_2 \times (p_1 + t_2 \times p_2) \\
q & \leftarrow & q_0 + t_2 \times (q_1 + t_2 \times q_2) \\
f & \leftarrow & (t \times p)/(q - t \times p) + 1/2
\end{array}\right\} \text{Approximation}
$$

    **return** $f \times 2^{k+1}$

$\rightarrow$ Crucial details of correctness proof:

- $t$ is close enough to $x - k \ln 2$: $x - k \times c_1$ is performed exactly
- No exceptional behaviour occurs in the approximation part
- The latter is a good enough approximation

# Correctness of floating-point algorithms

```
def cw_exp(x):
    if x < −746:
        return 0
    else if 710 < x:
        return +∞
    else:
        k  ← nearbyint(x × C)
        t  ← x − k × c1 − k × c2
        t2 ← t × t
        p  ← p0 + t2 × (p1 + t2 × p2)
        q  ← q0 + t2 × (q1 + t2 × q2)
        f  ← (t × p)/(q − t × p) + 1/2
        return f × 2^(k+1)
```

Lines $k$ and $t$: Reduction: $C \simeq 1/\ln 2$ and $c_1 + c_2 \simeq \ln 2$

Lines $t_2$ to $f$: Approximation

$\to$ Crucial details of correctness proof:

- $t$ is close enough to $x - k\ln 2$: $x - k \times c_1$ is performed exactly
- No exceptional behaviour occurs in the approximation part
- The latter is a good enough approximation

CORE-MATH logarithm:

```
static void cr_log_fast (double *h, double *l, int e, d64u64 v)
{
  uint64_t m = 0x10000000000000 + (v.u & 0xfffffffffffff);
  /* x = m/2^52 */
  /* if x > sqrt(2), we divide it by 2 to avoid cancellation */
  int c = m >= 0x16a09e667f3bcd;
  e += c; /* now -1074 <= e <= 1024 */
  static const double cy[] = {1.0, 0.5};
  static const uint64_t cm[] = {43, 44};
  int i = m >> cm[c];
  double y = v.f * cy[c];
  double r = (_INVERSE - OFFSET)[i];
  double z = __builtin_fma (r, y, -1.0); /* exact */


  /* evaluate P(z), for |z| < 0.00212097167968735 */
  double z2 = z * z;
  double p45 = __builtin_fma (P[5], z, P[4]);
  double p23 = __builtin_fma (P[3], z, P[2]);
...
```

# Interpretations of an abstract expression

Cody & Waite argument reduction: $e = (\mathbf{x} - \mathbf{k} \times \mathbf{c_1}) - \mathbf{k} \times \mathbf{c_2}$ (abstract)

- $[\![e]\!]_{\mathsf{flt}} = (\mathbf{x} -_{\mathbb{F}} \mathbf{k} \times_{\mathbb{F}} \mathbf{c_1}) -_{\mathbb{F}} \mathbf{k} \times_{\mathbb{F}} \mathbf{c_2}$ (IEEE 754: what the algorithm computes)
- $[\![e]\!]_{\mathsf{rnd}} = \circ(\circ(x - \circ(kc_1)) - \circ(kc_2))$ (rounded real numbers: what we would like to reason about)
- $[\![e]\!]_{\mathsf{exa}} = x - kc_1 - kc_2$ (infinite-precision real numbers: useful for formal proofs)

# Interpretations of an abstract expression

Cody & Waite argument reduction: $e = (\mathbf{x} - \mathbf{k} \times \mathbf{c_1}) - \mathbf{k} \times \mathbf{c_2}$ (abstract)

- $[\![e]\!]_{\mathsf{flt}} = (\mathbf{x} -_{\mathbb{F}} \mathbf{k} \times_{\mathbb{F}} \mathbf{c_1}) -_{\mathbb{F}} \mathbf{k} \times_{\mathbb{F}} \mathbf{c_2}$ (IEEE 754: what the algorithm computes)
- $[\![e]\!]_{\mathsf{rnd}} = \circ(\circ(x - \circ(kc_1)) - \circ(kc_2))$ (rounded real numbers: what we would like to reason about)
- $[\![e]\!]_{\mathsf{exa}} = x - kc_1 - kc_2$ (infinite-precision real numbers: useful for formal proofs)

Cody & Waite argument reduction: $e = (\mathbf{x} - \mathbf{k} \times \mathbf{c_1}) - \mathbf{k} \times \mathbf{c_2}$ (abstract)

- $[\![e]\!]_{\mathsf{flt}} = (\mathbf{x} -_{\mathbb{F}} \mathbf{k} \times_{\mathbb{F}} \mathbf{c_1}) -_{\mathbb{F}} \mathbf{k} \times_{\mathbb{F}} \mathbf{c_2}$ (IEEE 754: what the algorithm computes)
- $[\![e]\!]_{\mathsf{rnd}} = \circ(\circ(x - \circ(kc_1)) - \circ(kc_2))$ (rounded real numbers: what we would like to reason about)
- $[\![e]\!]_{\mathsf{exa}} = x - kc_1 - kc_2$ (infinite-precision real numbers: useful for formal proofs)

# Interpretations of an abstract expression

Cody & Waite argument reduction: $e = (\mathbf{x} - \mathbf{k} \times \mathbf{c_1}) - \mathbf{k} \times \mathbf{c_2}$ (abstract)

- $[\![e]\!]_{\mathsf{flt}} = (\mathbf{x} -_{\mathbb{F}} \mathbf{k} \times_{\mathbb{F}} \mathbf{c_1}) -_{\mathbb{F}} \mathbf{k} \times_{\mathbb{F}} \mathbf{c_2}$ (IEEE 754: what the algorithm computes)
- $[\![e]\!]_{\mathsf{rnd}} = \circ(\circ(x - \circ(kc_1)) - \circ(kc_2))$ (rounded real numbers: what we would like to reason about)
- $[\![e]\!]_{\mathsf{exa}} = x - kc_1 - kc_2$ (infinite-precision real numbers: useful for formal proofs)

# Correctness of an abstract expression

If $e$ is meant to approximate some ideal value $E$, we want to prove an assertion of the form:

$$[\![e]\!]_{\mathsf{flt}} \text{ is finite and } |[\![e]\!]_{\mathsf{flt}}/E - 1| \leq \varepsilon$$

- If $e$ meets certain conditions, we have $[\![e]\!]_{\mathsf{flt}} = [\![e]\!]_{\mathsf{rnd}}$, in which case it is sufficient to prove $|[\![e]\!]_{\mathsf{rnd}}/E - 1| \leq \varepsilon$
- $|[\![e]\!]_{\mathsf{rnd}}/E - 1| \leq \varepsilon$ is typically broken into $|[\![e]\!]_{\mathsf{rnd}}/[\![e]\!]_{\mathsf{exa}} - 1| \leq \varepsilon_1$ and $|[\![e]\!]_{\mathsf{exa}}/E - 1| \leq \varepsilon_2$

$\rightarrow$ **Goal**: Making this process less cumbersome for the user.

# Correctness of an abstract expression

If $e$ is meant to approximate some ideal value $E$, we want to prove an assertion of the form:

$$[\![e]\!]_{\mathsf{flt}} \text{ is finite and } |[\![e]\!]_{\mathsf{flt}}/E - 1| \leq \varepsilon$$

- If $e$ meets certain conditions, we have $[\![e]\!]_{\mathsf{flt}} = [\![e]\!]_{\mathsf{rnd}}$, in which case it is sufficient to prove $|[\![e]\!]_{\mathsf{rnd}}/E - 1| \leq \varepsilon$
- $|[\![e]\!]_{\mathsf{rnd}}/E - 1| \leq \varepsilon$ is typically broken into $|[\![e]\!]_{\mathsf{rnd}}/[\![e]\!]_{\mathsf{exa}} - 1| \leq \varepsilon_1$ and $|[\![e]\!]_{\mathsf{exa}}/E - 1| \leq \varepsilon_2$

$\rightarrow$ **Goal**: Making this process less cumbersome for the user.

# Correctness of an abstract expression

If $e$ is meant to approximate some ideal value $E$, we want to prove an assertion of the form:

$$[\![e]\!]_{\mathsf{flt}} \text{ is finite and } |[\![e]\!]_{\mathsf{flt}}/E - 1| \leq \varepsilon$$

- If $e$ meets certain conditions, we have $[\![e]\!]_{\mathsf{flt}} = [\![e]\!]_{\mathsf{rnd}}$, in which case it is sufficient to prove $|[\![e]\!]_{\mathsf{rnd}}/E - 1| \leq \varepsilon$
- $|[\![e]\!]_{\mathsf{rnd}}/E - 1| \leq \varepsilon$ is typically broken into $|[\![e]\!]_{\mathsf{rnd}}/[\![e]\!]_{\mathsf{exa}} - 1| \leq \varepsilon_1$ and $|[\![e]\!]_{\mathsf{exa}}/E - 1| \leq \varepsilon_2$

$\rightarrow$ **Goal**: Making this process less cumbersome for the user.

# Correctness of an abstract expression

If $e$ is meant to approximate some ideal value $E$, we want to prove an assertion of the form:

$$[\![e]\!]_{\mathsf{flt}} \text{ is finite and } |[\![e]\!]_{\mathsf{flt}}/E - 1| \leq \varepsilon$$

- If $e$ meets certain conditions, we have $[\![e]\!]_{\mathsf{flt}} = [\![e]\!]_{\mathsf{rnd}}$, in which case it is sufficient to prove $|[\![e]\!]_{\mathsf{rnd}}/E - 1| \leq \varepsilon$
- $|[\![e]\!]_{\mathsf{rnd}}/E - 1| \leq \varepsilon$ is typically broken into $|[\![e]\!]_{\mathsf{rnd}}/[\![e]\!]_{\mathsf{exa}} - 1| \leq \varepsilon_1$ and $|[\![e]\!]_{\mathsf{exa}}/E - 1| \leq \varepsilon_2$

$\rightarrow$ **Goal**: Making this process less cumbersome for the user.

# Contribution

- Abstract expressions:
  - Language and interpretations
  - Specification (relates $[\![.]\!]_{\text{flt}}$ and $[\![.]\!]_{\text{rnd}}$)
- Tools for the Coq proof assistant

# Expressions and interpretations

# Language of expressions

```
Let k = NearbyInt (Op MUL (Var x) InvLog2)
Let t = Op SUB
 (OpExact SUB (Var x) (OpExact MUL (Var k) Log2h))
 (Op MUL (Var k) Log2l)
```

$\longleftrightarrow$ $\begin{array}{lll} k & \leftarrow & \text{nearbyint}(x \times C) \\ t & \leftarrow & x - k \times c_1 - k \times c_2 \end{array}$

$\rightarrow$ Supported operations: $+$, $-$, $\times$, $/$, $\sqrt{}$, $\lfloor . \rceil$, FMA, etc.

$\rightarrow$ Exact results:

- $[\![u +_{\text{exact}} v]\!]_{\text{flt}} = [\![u]\!]_{\text{flt}} +_{\mathbb{F}} [\![v]\!]_{\text{flt}} = [\![u + v]\!]_{\text{flt}}$
- $[\![u +_{\text{exact}} v]\!]_{\text{rnd}} = [\![u]\!]_{\text{rnd}} + [\![v]\!]_{\text{rnd}} \neq [\![u + v]\!]_{\text{rnd}}$

# Language of expressions

```
Let k = NearbyInt (Op MUL (Var x) InvLog2)
Let t = Op SUB
 (OpExact SUB (Var x) (OpExact MUL (Var k) Log2h))
 (Op MUL (Var k) Log2l)
```

$\longleftrightarrow$

$$\begin{array}{lcl} k & \leftarrow & \text{nearbyint}(x \times C) \\ t & \leftarrow & x - k \times c_1 - k \times c_2 \end{array}$$

$\rightarrow$ Supported operations: $+$, $-$, $\times$, $/$, $\sqrt{}$, $\lfloor . \rceil$, FMA, etc.

$\rightarrow$ Exact results:

- $[\![u +_{\text{exact}} v]\!]_{\text{flt}} = [\![u]\!]_{\text{flt}} +_{\mathbb{F}} [\![v]\!]_{\text{flt}} = [\![u + v]\!]_{\text{flt}}$
- $[\![u +_{\text{exact}} v]\!]_{\text{rnd}} = [\![u]\!]_{\text{rnd}} + [\![v]\!]_{\text{rnd}} \neq [\![u + v]\!]_{\text{rnd}}$

# Well-behaved expressions

First we want to prove $[\![e]\!]_{\text{flt}}$ is finite and represents $[\![e]\!]_{\text{rnd}}$.

We define (by induction) a predicate $\mathrm{WB}$ on expressions such that if $\mathrm{WB}(e)$ holds then $e$ is well-behaved.

- $\mathrm{WB}(u/v) \triangleq \begin{cases} \quad \mathrm{WB}(u) \wedge \mathrm{WB}(v) \\ \wedge \quad |[\![v]\!]_{\text{rnd}}| \neq 0 \\ \wedge \quad |\circ([\![u]\!]_{\text{rnd}}/[\![v]\!]_{\text{rnd}})| \leq \Omega \end{cases}$ (no division by 0) (no overflow)

- $\mathrm{WB}(u +_{\text{exact}} v) \triangleq \begin{cases} \quad \mathrm{WB}(u) \wedge \mathrm{WB}(v) \\ \wedge \quad \circ([\![u]\!]_{\text{rnd}} + [\![v]\!]_{\text{rnd}}) = [\![u]\!]_{\text{rnd}} + [\![v]\!]_{\text{rnd}} \\ \wedge \quad |[\![u]\!]_{\text{rnd}} + [\![v]\!]_{\text{rnd}}| \leq \Omega \end{cases}$ (produces exact result) (no overflow)

**Correspondence theorem**

$$\mathrm{WB}(e) \Rightarrow [\![e]\!]_{\text{flt}} \text{ finite} \wedge [\![e]\!]_{\text{flt}} = [\![e]\!]_{\text{rnd}}$$

# Tools for the Coq proof assistant

# Tools for the Coq proof assistant

- Process a goal about $[\![e]\!]_{\mathsf{flt}}$ and apply correspondence theorem to obtain a goal about $[\![e]\!]_{\mathsf{rnd}}$, yields a $\mathrm{WB}(e)$ goal not ideal to prove by hand
- Try to prove automatically all conjuncts of $\mathrm{WB}(e)$
- Facilitate asserting a property on a subexpression (c.f. Cody & Waite)

$$A(t) := |t| \leqslant \frac{\ln 2}{2} \wedge \cdots$$

✓ ✓ ✓

CoqInterval, Gappa

$$|[\![e]\!]_{\mathsf{rnd}} / \exp(t) - 1| \leqslant \varepsilon$$

by hand, Gappa

by hand, Gappa

simplify_wb

$$\circ(x - \circ(kc_1)) = x - kc_1$$

$$\mathrm{WB}(e) \wedge |[\![e]\!]_{\mathsf{rnd}} / \exp(t) - 1| \leqslant \varepsilon$$

simplify_wb

intros

$$\mathrm{WB}(t) \qquad \mathrm{WB}(t) \Rightarrow A([\![t]\!]_{\mathsf{rnd}}) \qquad \forall (t : \mathbb{R}),\ A(t) \Rightarrow \mathrm{WB}(e) \wedge |[\![e]\!]_{\mathsf{rnd}} / \exp(t) - 1| \leqslant \varepsilon$$

assert_subexpr

$$\mathrm{WB}(\texttt{let}\ t := \dots\ \texttt{in}\ e) \wedge |[\![\texttt{let}\ t := \dots\ \texttt{in}\ e]\!]_{\mathsf{rnd}} / \exp(t) - 1| \leqslant \varepsilon$$

correspondence theorem

$$[\![\texttt{let}\ t := \dots\ \texttt{in}\ e]\!]_{\mathsf{flt}} \text{ is finite} \wedge |[\![\texttt{let}\ t := \dots\ \texttt{in}\ e]\!]_{\mathsf{flt}} / \exp(t) - 1| \leqslant \varepsilon$$

# Automating proof of $\mathrm{WB}(e)$

MITSUBISHI ELECTRIC
*Changes for the Better*

Automatic tools use interval arithmetic (c.f. Gappa, CoqInterval).
CoqInterval can perform finer interval arithmetic using Taylor models.
Gappa supports roundings and makes use of floating-point theorems.

- $|\circ(u+v)| \leqslant \Omega$ can be proven using naïve interval arithmetic
- $v \neq 0$ can be proven using interval arithmetic with Taylor models
- $\circ(x+y) = x+y$ generally cannot be proven using just interval arithmetic

$\rightarrow$ Added support for roundings in CoqInterval's naïve and Taylor-based provers.

 

# Automating proof of $\mathrm{WB}(e)$

Automatic tools use interval arithmetic (c.f. Gappa, CoqInterval).
CoqInterval can perform finer interval arithmetic using Taylor models.
Gappa supports roundings and makes use of floating-point theorems.

- $|\circ(u+v)| \leqslant \Omega$ can be proven using naïve interval arithmetic
- $v \neq 0$ can be proven using interval arithmetic with Taylor models
- $\circ(x+y) = x+y$ generally cannot be proven using just interval arithmetic

$\rightarrow$ Added support for roundings in CoqInterval's naïve and Taylor-based provers.

# Automating proof of $\mathrm{WB}(e)$

Automatic tools use interval arithmetic (c.f. Gappa, CoqInterval).
CoqInterval can perform finer interval arithmetic using Taylor models.
Gappa supports roundings and makes use of floating-point theorems.

- $|\circ(u+v)| \leqslant \Omega$ can be proven using naïve interval arithmetic
- $v \neq 0$ can be proven using interval arithmetic with Taylor models
- $\circ(x+y) = x+y$ generally cannot be proven using just interval arithmetic

$\rightarrow$ Added support for roundings in CoqInterval's naïve and Taylor-based provers.

# Conclusion

# Summary

We built a Coq tool for facilitating proofs about floating-point approximations.
Available in CoqInterval: `https://coqinterval.gitlabpages.inria.fr/`

Correctness of the polynomial approximation of CORE-MATH 64-bit logarithm:

$$-0.00203 \leqslant z \leqslant 0.00212 \to [\![P(z)]\!]_{\mathsf{flt}} \text{ finite} \wedge |[\![P(z)]\!]_{\mathsf{flt}} - (\ln(1+z) - z)| \leq 2^{-68.72}$$

$\Rightarrow$ Proved in 7 lines of Coq.

Tested examples are available here: `https://gitlab.inria.fr/pgeneaud/examples`
The CORE-MATH project: `https://core-math.gitlabpages.inria.fr/`

# Summary

We built a Coq tool for facilitating proofs about floating-point approximations.
Available in CoqInterval: `https://coqinterval.gitlabpages.inria.fr/`

Correctness of the polynomial approximation of CORE-MATH 64-bit logarithm:

$$-0.00203 \leqslant z \leqslant 0.00212 \to [\![P(z)]\!]_{\mathsf{flt}} \text{ finite} \wedge |[\![P(z)]\!]_{\mathsf{flt}} - (\ln(1+z) - z)| \leq 2^{-68.72}$$

$\Rightarrow$ Proved in 7 lines of Coq.

Tested examples are available here: `https://gitlab.inria.fr/pgeneaud/examples`
The CORE-MATH project: `https://core-math.gitlabpages.inria.fr/`

# Perspectives

→ Full correctness of the CORE-MATH logarithm: macro-operations (FastTwoSum), tables of constants, support for control flow (language of instructions)

→ Support for higher-level procedures (argument reduction, polynomial/rational approximation, etc.)

→ Working directly with C programs (translation from C to the language of expressions, and back)

LMF: `https://lmf.cnrs.fr`