Floating-point arithmetic in the Coq system

Guillaume Melquiond

Mathematical Components
INRIA–Microsoft Research

2008-07-08
While certifying an algorithm for preventing airplane collision, Carreño and Muñoz had to formally prove:

\[ \frac{3\pi}{180} \leq \frac{g}{v} \cdot \tan \left( \frac{35\pi}{180} \right) \]

In other words, \( g = 9.8 \ m/s^2 \) and \( v = 250 \cdot \frac{514}{1000} \ m/s \).
Verifying an equality, an example

The inequality is trivially true since

\[
\frac{3\pi}{180} \approx 0.052 \quad \text{and} \quad \frac{g}{v} \cdot \tan \left( \frac{35\pi}{180} \right) \approx 0.053.
\]
Verifying an equality, an example

The inequality is trivially true since
\[
\frac{3\pi}{180} \approx 0.052 \quad \text{and} \quad \frac{g}{v} \cdot \tan \left( \frac{35\pi}{180} \right) \approx 0.053.
\]

Proved in PVS by Lester and Muñoz thanks to interval computations on rational bounds.
Formal proofs and floating-point computations

Kepler’s conjecture on 3D sphere packing.
Kepler’s conjecture on 3D sphere packing.

- Hales splits the problem into **thousands** of configurations. Each configuration is described by a system with 6 variables, then solved by **interval computations** on **floating-point bounds**.

  **Confidence** of the reviewers: only 95% due to the computationally-intensive proof.
Kepler’s conjecture on 3D sphere packing.

- Hales splits the problem into thousands of configurations. Each configuration is described by a system with 6 variables, then solved by interval computations on floating-point bounds.

  Confidence of the reviewers: only 95% due to the computationally-intensive proof.

- Hales now aims to perform a formal proof in order to achieve 100% confidence.

But what about computations?
### Natural numbers:

```ocaml
type nat = O | S of nat  (* 5 = SSSSSO *)
let rec plus x y =
  match x with
  | O -> y
  | S x' -> plus x' (S y)
```

### Computations and proofs

Proof by computation:

1. Goal (plus 3 5) = (plus 5 3). trivial.
Computations and proofs

Natural numbers:

\[
\text{type nat } = \text{ O | S of nat} \quad (\star \ 5 = \text{SSSSSO} \ \star) \\
\text{let rec plus x y } = \\
\text{ match x with} \\
\quad | \text{ O } \rightarrow \ y \\
\quad | \text{ S x' } \rightarrow \text{ plus x' (S y)}
\]

Proof by computation:

\[
\text{Goal (plus 3 5) } = \text{(plus 5 3). trivial.}
\]
Computations and proofs

Natural numbers:

\[
\text{type \ nat = O | S of nat} \quad (\ast \ 5 = \text{SSSSSO} \ \ast)
\]

\[
\text{let rec plus x y =}
\]

\[
\begin{align*}
\text{match x with} & \\
| \text{O} & \to \ y & \\
| \text{S x'} & \to \ \text{plus x'} (\text{S y}) & \\
\end{align*}
\]

Proof by computation:

\[
\text{Goal (plus 3 5) = (plus 5 3). trivial.}
\]

Standard positive integers:

\[
\text{type \ pos = H | B0 of pos | B1 of pos} \quad (\ast \ 26 = \text{B0B1B0B1H} \ \ast)
\]
Floating-point arithmetic in the Coq system

**Goal:** Implement an efficient arithmetic inside a formal system.

1. Number representations
2. Basic arithmetic operators
3. Elementary functions
4. Conclusion
Outline

1. Number representations
   - FP numbers
   - Libraries
   - Modules
   - Fast integers

2. Basic arithmetic operators

3. Elementary functions

4. Conclusion
Definitions (Floating-point number)

Floating-point numbers in radix $\beta$:

- Pairs of integers $(m, e) \in \mathbb{Z}^2$ interpreted as $m \cdot \beta^e \in \mathbb{R}$.
- Not-a-Number $\perp$ for exceptional behavior: $\frac{1}{0}$, $\sqrt{-1}$, etc.
**Definition (Floating-point number)**

Floating-point numbers in radix $\beta$:
- Pairs of integers $(m, e) \in \mathbb{Z}^2$ interpreted as $m \cdot \beta^e \in \mathbb{R}$.
- **Not-a-Number** $\perp$ for exceptional behavior: $\frac{1}{0}$, $\sqrt{-1}$, etc.

Unbounded exponent range $\implies$ no overflow nor underflow.
No infinities nor signed zeros.
Rounding

Some real numbers are not representable as floating-point values. E.g. \( \frac{1}{3} \) for \( \beta = 2 \), \( \sqrt{2} \) for any \( \beta \).
Some real numbers are not representable as floating-point values. E.g. $\frac{1}{3}$ for $\beta = 2$, $\sqrt{2}$ for any $\beta$.

**Solution:** Use a FP value close to the real number instead. Uniquely defined according to a *precision* and a *rounding direction*.

**Example:** In radix 10 and precision 4, $\pi$ is rounded to:
- $3141 \cdot 10^{-3}$ when rounding toward $-\infty$ or zero,
- $3142 \cdot 10^{-3}$ when rounding toward $+\infty$ or to nearest.
Rounding

Some real numbers are not representable as floating-point values. E.g. $\frac{1}{3}$ for $\beta = 2$, $\sqrt{2}$ for any $\beta$.

Solution: Use a FP value close to the real number instead. Uniquely defined according to a precision and a rounding direction.

Example: In radix 10 and precision 4, $\pi$ is rounded to:
- $3141 \cdot 10^{-3}$ when rounding toward $-\infty$ or zero,
- $3142 \cdot 10^{-3}$ when rounding toward $+\infty$ or to nearest.

Rounding direction and precision are specified for each operation:

$$\text{Fadd } \text{rnd}_{\text{UP}} 500 : \mathbb{F}_\beta \rightarrow \mathbb{F}_\beta \rightarrow \mathbb{F}_\beta.$$
Existing floating-point libraries

A few libraries on floating-point numbers:

- Moore, Lynch, and Kaufmann’s library in ACL2.
- Daumas, Rideau, and Théry’s library in Coq.
- Harrison’s library in HOL light.
Existing floating-point libraries

A few libraries on floating-point numbers:

- Moore, Lynch, and Kaufmann’s library in ACL2.
- Daumas, Rideau, and Théry’s library in Coq.
- Harrison’s library in HOL light.

Definition of rounding to $-\infty$:

$$\text{downward}(\text{format}, f, x) =$$

$$f \in \text{format} \land f \leq x \land \forall g \in \text{format}, g \leq x \Rightarrow g \leq f.$$
Existing floating-point libraries

A few libraries on floating-point numbers:
- Moore, Lynch, and Kaufmann’s library in ACL2.
- Daumas, Rideau, and Théry’s library in Coq.
- Harrison’s library in HOL light.

Definition of rounding to $-\infty$:
\[
downward(format, f, x) =
\]
\[
f \in format \land f \leq x \land \forall g \in format, g \leq x \Rightarrow g \leq f.
\]

Rounding predicates:
- useful for certifying floating-point programs,
- useless for computing with floating-point numbers.
Modular design

The library provides a generic multi-radix implementation based on Coq’s standard integers. But

- an optimized radix-2 arithmetic is better for computations, and/or
- the user may trust the integer arithmetic units of the CPU.
The library provides a **generic** multi-radix implementation based on Coq’s **standard** integers. But

- an optimized radix-2 arithmetic is better for computations, and/or
- the user may trust the integer arithmetic units of the CPU.

Using Coq’s **module system**:

- an interface (module type) describes floating-point operations,
- several implementations (modules) provides them:
  - `GenericFloat α`: any radix $\alpha$, standard integers,
  - `SpecificFloat StdZRadix2`: radix 2, standard integers,
  - `SpecificFloat BigIntRadix2`: radix 2, fast integers.
Fast integers: Native 31-bit integers

Arithmetic on 31-bit integers has been formally defined in Coq.

- Addition returns an integer and a carry.
- Multiplication returns high and low parts of the result.
Fast integers: Native 31-bit integers

Arithmetic on 31-bit integers has been formally defined in Coq.
- Addition returns an integer and a carry.
- Multiplication returns high and low parts of the result.

Spiwack has modified the virtual machine to improve performance:
- Closed-term integers are compressed into machine words (Ocaml-like encoding).
- Addition, division, rotation, leading-zero count, etc are delegated to the CPU.
Grégoire and Théry have defined an integer type as a binary tree with int31 leaves.

\[ a = \sum_{k} a_k \cdot 2^{31 \cdot k} \]

- Logarithmic complexity for accessing digits.
- Divide-and-conquer algorithms, e.g. Karatsuba’s multiplication.
The set of floating-point numbers $\mathbb{F}_\beta - \{\bot\}$ is a ring for the addition and multiplication on real numbers.

**FP addition** and **multiplication** first compute the exact value:

- $(m_1, e_1) + (m_2, e_2) \rightarrow (m_1 \cdot \beta^{e_1-e_2} + m_2, e_2)$ for $e_2 \leq e_1$.
- $(m_1, e_1) \times (m_2, e_2) \rightarrow (m_1 \cdot m_2, e_1 + e_2)$.

This value is then **rounded** to the target precision.
Division and square root

For the division of \((m_1, e_1)\) by \((m_2, e_2)\):

1. Increase the width of the input: \((m'_1, e'_1) = (m_1 \cdot \beta^k, e_1 - k)\), so that \(\lfloor m'_1 / m_2 \rfloor\) has at least \(p\) digits.

2. Perform an integer operation on mantissa: \((\lfloor m'_1 / m_2 \rfloor, e'_1 - e_2)\).

3. Round the number to \(p\) digits, according to the difference \((m'_1 - m_2 \cdot \lfloor m'_1 / m_2 \rfloor) - m_2 / 2\).
Division and square root

For the division of \((m_1, e_1)\) by \((m_2, e_2)\):

1. Increase the width of the input: \((m'_1, e'_1) = (m_1 \cdot \beta^k, e_1 - k)\), so that \(\lfloor m'_1/m_2 \rfloor\) has at least \(p\) digits.
2. Perform an integer operation on mantissa: \((\lfloor m'_1/m_2 \rfloor, e'_1 - e_2)\).
3. Round the number to \(p\) digits, according to the difference \((m'_1 - m_2 \cdot \lfloor m'_1/m_2 \rfloor) - m_2/2\).

Similarly for the square root of \((m, e)\):

2. Perform an integer operation on mantissa: \((\lfloor \sqrt{m'} \rfloor, e'/2)\).
3. Round according to \((m' - \lfloor \sqrt{m'} \rfloor^2) - (2 \cdot \lfloor \sqrt{m'} \rfloor - 1)/2\).
Definition (Interval)

An interval is a **closed connected subset** of the real numbers, or \( \bot \).

Some subsets can be represented as **pairs** of FP numbers \( \mathbb{I} = \mathbb{R}^2 \):

- \( x \in [\bot, \bot] \Rightarrow x \in \mathbb{R} \),
- \( x \in [\bot, b] \Rightarrow x \in \mathbb{R} \land x \leq b \),
- \( x \in [a, \bot] \Rightarrow x \in \mathbb{R} \land a \leq x \),
- \( x \in [a, b] \Rightarrow x \in \mathbb{R} \land a \leq x \leq b \),
- \( x \in \bot \) holds for any \( x \in \mathbb{R} \cup \{\bot\} \).
Interval evaluations can serve as **proofs** of bounds, when they satisfy the **containment property**:

\[ x \in l_x \land y \in l_y \implies x \diamond y \in l_x \diamond l_y \]

for \( \diamond \in \{+,-,\times,\div\} \). Also for unary functions: \( \sqrt{-}, \ldots \)
Interval arithmetic

Interval evaluations can serve as proofs of bounds, when they satisfy the containment property:

\[ x \in l_x \land y \in l_y \implies x \circ y \in l_x \circ l_y \]

for \( \circ \in \{+, -, \times, \div\} \). Also for unary functions: \( \sqrt{\ldots} \).

Arithmetic operations on intervals with FP bounds:

- \([a, b] + [c, d] = [\nabla (a + c), \triangle (b + d)]\),
- \([a, b] - [c, d] = [\nabla (a - d), \triangle (b - c)]\),
- \(\ldots\)
Outline

1. Number representations
2. Basic arithmetic operators

3. Elementary functions
   - Rounding
   - Computation
   - Performances

4. Conclusion
Elementary functions: cos, arctan, exp, etc.
Difficulty: Their mathematical result $y$ cannot generally be split into representable floating-point numbers.
Elementary functions

Elementary functions: cos, arctan, exp, etc.

Difficulty: Their mathematical result $y$ cannot generally be split into representable floating-point numbers.

No correct rounding for elementary functions:

- They return an interval enclosing the mathematical value.
- Precision is only an hint for the intermediate computations.
Computing elementary functions

**Argument reduction** until the input is smaller than $\frac{1}{2}$:

\[
\begin{align*}
\cos x &= 2 \cdot (\cos \frac{x}{2})^2 - 1 \\
\text{sign}(\sin x) &= \text{sign}(\sin \frac{x}{2}) \cdot \text{sign}(\cos \frac{x}{2}) \\
\sin x &= \text{sign}(\sin x) \cdot \sqrt{1 - (\cos x)^2} \quad \text{(Only for } |x| \leq 2^{20}.\text{)}
\end{align*}
\]
Computing elementary functions

**Argument reduction** until the input is smaller than $\frac{1}{2}$:

\[
\begin{align*}
\cos x &= 2 \cdot (\cos \frac{x}{2})^2 - 1 \\
\text{sign}(\sin x) &= \text{sign}(\sin \frac{x}{2}) \cdot \text{sign}(\cos \frac{x}{2}) \\
\sin x &= \text{sign}(\sin x) \cdot \sqrt{1 - (\cos x)^2} \quad \text{(Only for } |x| \leq 2^{20}.)
\end{align*}
\]

**Evaluation of series** until the truncated part is negligible:

\[
\sin x = x \cdot (1 - \frac{x^2}{6} + \cdots + \epsilon) \quad \text{with } |\epsilon| \leq \frac{x^{2n}}{(2n+1)!} \leq \beta^{-p}
\]

by interval arithmetic:

\[
\sin x \in X \cdot (1 - \frac{x^2}{6} + \cdots + [\pm \frac{|X|^{2n}}{(2n+1)!}]) \quad \text{for } x \in X.
\]
Rough performances

Tests against MPFR data sets (53-bit radix-2 mantissas).

**Strategy for correct rounding:**
- First compute with an intermediate precision $p = 63$.
- If the bounds are not rounded to the same 53-bit FP number, try again with an intermediate precision $p \times 1.5$, and so on.
Tests against MPFR data sets (53-bit radix-2 mantissas).

**Strategy for correct rounding:**
First compute with an intermediate precision $p = 63$.
If the bounds are not rounded to the same 53-bit FP number, try again with an intermediate precision $p \times 1.5$, and so on.

**Average delay for evaluating a function:** (processor cycles)

<table>
<thead>
<tr>
<th>Function</th>
<th>Tested values</th>
<th>Coq fast</th>
<th>Coq std</th>
<th>MPFR</th>
</tr>
</thead>
<tbody>
<tr>
<td>arctan</td>
<td>870 (100%)</td>
<td>$20 \cdot 10^6$</td>
<td>$10 \cdot 10^7$</td>
<td>$57 \cdot 10^3$</td>
</tr>
<tr>
<td>cos</td>
<td>1289 (91%)</td>
<td>$27 \cdot 10^6$</td>
<td>$20 \cdot 10^7$</td>
<td>$21 \cdot 10^3$</td>
</tr>
<tr>
<td>sin</td>
<td>1120 (89%)</td>
<td>$32 \cdot 10^6$</td>
<td>$81 \cdot 10^7$</td>
<td>$21 \cdot 10^3$</td>
</tr>
<tr>
<td>tan</td>
<td>1297 (81%)</td>
<td>$36 \cdot 10^6$</td>
<td>$83 \cdot 10^7$</td>
<td>$22 \cdot 10^3$</td>
</tr>
</tbody>
</table>
Outline

1. Number representations
2. Basic arithmetic operators
3. Elementary functions
4. Conclusion
   - Realizations
   - Application
Realizations

Pure library. No modifications to Coq.

11400 lines of Coq:

- floating-point arithmetic: 4600 lines,
- interval arithmetic: 2000 lines,
- tactics: 2500 lines.

↑ Automatic solver for inequalities on real-values expressions.
Verifying inequalities, another example

Global positioning requires knowing the local radius of Earth:

\[ r_p(\phi) = \frac{a}{\sqrt{1 + (1 - f)^2 \cdot \tan^2 \phi}} \]

This can be approximated by a degree-5 polynomial \( P \) with single-precision coefficients: \( P(\phi^2_m - \phi^2) = \tilde{r}_p(\phi) \).

\[ P(x) = \frac{4439091}{4} + x \cdot \left( \frac{9023647}{4} + x \cdot \left( \cdots \frac{6661427}{131072} \right) \right) \]

**Goal:** Prove \( \left| \frac{r_p(\phi) - \tilde{r}_p(\phi)}{r_p(\phi)} \right| \leq 23 \cdot 2^{-24} \) when \( 0 \leq \phi \leq \phi_m = \frac{715}{512} \).
Verifying inequalities, another example

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$. 
Verifying inequalities, another example

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.


   An oracle chooses the best sub-intervals and it generates one PVS script per sub-interval.

Verification: several hours on a 48-core parallel computer.
Verifying inequalities, another example

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.

1. Daumas, Melquiond, and Muñoz, in PVS (2005). An oracle chooses the best sub-intervals and it generates one PVS script per sub-interval. Verification: several hours on a 48-core parallel computer.

2. In Coq (2008), a few minutes on a laptop computer.
Verifying inequalities, another example

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.

   An oracle chooses the best sub-intervals and it generates one PVS script per sub-interval.
   Verification: several hours on a 48-core parallel computer.

2. In Coq (2008), a few minutes on a laptop computer.
   Differences: No oracle, but floating-point numbers.
Verifying inequalities, another example

Require Import Reals.
Require Import tactics.
Open Local Scope R_scope.

Definition a := 6378137.
Definition f := 1000000000/298257223563.
Definition umf2 := (1 - f)^2.
Definition max := 715/512.
Definition rp phi := a / sqrt (1 + umf2 * (tan phi)^2).

Definition arp phi :=
  let x := max^2 - phi^2 in
  4439091/4 + x * (9023647/4 + x * (13868737/64 + x * (13233647/2048 + x * (-1898597/16384 + x * (-6661427/131072)))).

Goal forall phi, 0 <= phi <= max -> Rabs ((rp phi - arp phi) / rp phi) <= 23/16777216.
Proof.
unfold rp, arp, umf2, a, f, max. intros.
Time interval with (i_bisect_diff phi, i_nocheck). (* 4s *)
Time Qed. (* 96s *)

Tactic parameters:
bisection and order-1 evaluation on \(\phi\),
check delayed at Qed time,
minimal relative width of intervals: \(2^{-15}\),
precision: 30 bits.
Questions?

Mail: guillaume.melquiond@inria.fr
Web: http://www.msr-inria.inria.fr/soft/coq-interval/