

\mathbb{R} and \mathbb{F}

Sylvie Boldo

INRIA – ProVal

January 19th, 2012



Advertising

- I belong to the ProVal team (Saclay, south of Paris).
- My core research is about proofs of numerical programs.
- I am linked to the 2-36-1 course (Proofs of Programs) of Claude Marché and Guillaume Melquiond.

Outline

- 1 \mathbb{R} : formalization
- 2 Real-life \mathbb{R} : floating-point numbers
- 3 \mathbb{F} : formalization
- 4 Proofs about \mathbb{F}

What is \mathbb{R} ?

How to **define** \mathbb{R} ?

- infinite expansions

What is \mathbb{R} ?

How to define \mathbb{R} ?

- infinite expansions
- Dedekind cut

What is \mathbb{R} ?

How to define \mathbb{R} ?

- infinite expansions
- Dedekind cut
- Cauchy sequences

What is \mathbb{R} ?

How to define \mathbb{R} ?

- infinite expansions
- Dedekind cut
- Cauchy sequences
- other mathematical constructions

What is \mathbb{R} ?

How to **define** \mathbb{R} ?

- infinite expansions
- Dedekind cut
- Cauchy sequences
- other mathematical constructions

- axiomatization!

Construction vs axiomatization

Construction

- ⊕ usual mathematical method

Axiomatization

Construction vs axiomatization

Construction

- ⊕ usual mathematical method
- ⊕ safety

Axiomatization

Construction vs axiomatization

Construction

- ⊕ usual mathematical method
- ⊕ safety
- ⊕ calculable (most of the time)

Axiomatization

Construction vs axiomatization

Construction

- ⊕ usual mathematical method
- ⊕ safety
- ⊕ calculable (most of the time)
- ⊖ **big effort** for the formalization and the proofs

Axiomatization

Construction vs axiomatization

Construction

- ⊕ usual mathematical method
- ⊕ safety
- ⊕ calculable (most of the time)
- ⊖ **big effort** for the formalization and the proofs

Axiomatization

- ⊕ very fast

Construction vs axiomatization

Construction

- ⊕ usual mathematical method
- ⊕ safety
- ⊕ calculable (most of the time)
- ⊖ **big effort** for the formalization and the proofs

Axiomatization

- ⊕ very fast
- ⊕ easier and faster to prove interesting facts

Construction vs axiomatization

Construction

- ⊕ usual mathematical method
- ⊕ safety
- ⊕ calculable (most of the time)
- ⊖ **big effort** for the formalization and the proofs

Axiomatization

- ⊕ very fast
- ⊕ easier and faster to prove interesting facts
- ⊖ must be very **careful about axioms**
⇒ your logic might become incoherent

Classic vs intuitionist

Intuitionist

- ⊕ extract programs

Classic

Classic vs intuitionist

Intuitionist

- ⊕ extract programs
- ⊖ no way to prove given theorems (Rolle theorem)

Classic

Classic vs intuitionist

Intuitionist

- ⊕ extract programs
- ⊖ no way to prove given theorems (Rolle theorem)
- ⊖ we may prove two real numbers are different (*apart*, $\#$),
but we **cannot decide if they are equal**

Classic

Classic vs intuitionist

Intuitionist

- ⊕ extract programs
- ⊖ no way to prove given theorems (Rolle theorem)
- ⊖ we may prove two real numbers are different (*apart*, $\#$),
but we **cannot decide if they are equal**

Classic

- ⊕ very fast

Classic vs intuitionist

Intuitionist

- ⊕ extract programs
- ⊖ no way to prove given theorems (Rolle theorem)
- ⊖ we may prove two real numbers are different (*apart*, $\#$), but we **cannot decide if they are equal**

Classic

- ⊕ very fast
- ⊕ easier and faster to prove interesting facts

Classic vs intuitionist

Intuitionist

- ⊕ extract programs
- ⊖ no way to prove given theorems (Rolle theorem)
- ⊖ we may prove two real numbers are different (*apart*, $\#$), but we **cannot decide if they are equal**

Classic

- ⊕ very fast
- ⊕ easier and faster to prove interesting facts
- ⊖ must be very careful about axioms
⇒ your logic might become incoherent

Formalizations of \mathbb{R}

		intuitionist	classic
construction	+	proved extractible	proved
	-	long complex	long
axiomatization	+	fast \pm extractible	fast easy
	-	unproved complex	unproved not extractible

In Coq, several definitions are available including:

- the standard library: axiomatization & classic
- C-Corn/Math Classes (Nijmegen):
construction (Cauchy) & intuitionist

About the standard library \mathbb{R}

\mathbb{R} a **Parameter** of type **Set**. We assume the set and functions having the expected behavior but we do not construct anything.

Parameter $\mathbb{R} : \mathbf{Set}$.

Parameter $\mathbb{R}0 : \mathbb{R}$.

Parameter $\mathbb{R}1 : \mathbb{R}$.

Parameter $\mathbb{R}\text{plus} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}$.

Parameter $\mathbb{R}\text{mult} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}$.

Parameter $\mathbb{R}\text{opp} : \mathbb{R} \rightarrow \mathbb{R}$.

Parameter $\mathbb{R}\text{inv} : \mathbb{R} \rightarrow \mathbb{R}$.

Parameter $\mathbb{R}\text{lt} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbf{Prop}$.

Parameter $\text{up} : \mathbb{R} \rightarrow \mathbb{Z}$.

Axioms of the standard library \mathbb{R}

Here are the first axioms about the operations $+$, \times , $-$ (unary), $/$ (unary).

Axiom `Rplus_comm` : **forall** `r1 r2:R`, $r1 + r2 = r2 + r1$.

Axiom `Rplus_assoc` : **forall** `r1 r2 r3:R`,
 $r1 + r2 + r3 = r1 + (r2 + r3)$.

Axiom `Rplus_opp_r` : **forall** `r:R`, $r + - r = 0$.

Axiom `Rplus_0_l` : **forall** `r:R`, $0 + r = r$.

Axiom `Rmult_comm` : **forall** `r1 r2:R`, $r1 * r2 = r2 * r1$.

Axiom `Rmult_assoc` : **forall** `r1 r2 r3:R`,
 $r1 * r2 * r3 = r1 * (r2 * r3)$.

Axiom `Rinv_l` : **forall** `r:R`, $r \triangleleft 0 \rightarrow / r * r = 1$.

Axiom `Rmult_1_l` : **forall** `r:R`, $1 * r = r$.

Axiom `Rmult_plus_distr_l` : **forall** `r1 r2 r3:R`,
 $r1 * (r2 + r3) = r1 * r2 + r1 * r3$.

Axioms of the standard library \mathbb{R}

To prevent all the real numbers to be zero, we assume that $0 \neq 1$:

Axiom `R1_neq_R0` : $1 \neq 0$.

We assume that the order (and equality) are decidable. We may decide in a function the sign of a real number:

Axiom `total_order_T` : **forall** `r1 r2` : \mathbb{R} ,
 $\{r1 < r2\} + \{r1 = r2\} + \{r1 > r2\}$.

Axioms of the standard library \mathbb{R}

A few axioms on the order:

Axiom Rlt_asym : **forall** r1 r2:R, r1 < r2 \rightarrow \sim r2 < r1 .

Axiom Rlt_trans : **forall** r1 r2 r3:R,
r1 < r2 \rightarrow r2 < r3 \rightarrow r1 < r3 .

A few axioms on the order and the operations:

Axiom Rplus_lt_compat_l : **forall** r r1 r2:R,
r1 < r2 \rightarrow r + r1 < r + r2 .

Axiom Rmult_lt_compat_l : **forall** r r1 r2:R,
0 < r \rightarrow r1 < r2 \rightarrow r * r1 < r * r2 .

About the standard library \mathbb{R}

And we define \leq as being either $<$ or $=$:

Definition $\text{Rle} (r1\ r2 : \mathbb{R}) : \mathbf{Prop} := r1 < r2 \ \vee \ r1 = r2.$

And we define $\text{INR} (\mathbb{N} \rightarrow \mathbb{R})$ and $\text{IZR} (\mathbb{Z} \rightarrow \mathbb{R})$ with $0 \rightarrow R0$, $1 \rightarrow R1$, etc.

The real field is archimedean: we can find an integer greater than any real. Here, we assume it is the smallest one:

Axiom $\text{archimed} : \mathbf{forall} \ r : \mathbb{R},$
 $\text{IZR} (\text{up } r) > r \ \wedge \ \text{IZR} (\text{up } r) - r \leq 1.$

Axioms of the standard library \mathbb{R}

And \mathbb{R} is complete: any non empty set has a bigger element:

Definition `is_upper_bound (E:R → Prop) (m:R) :=
forall x:R, E x → x <= m.`

Definition `bound (E:R → Prop) :=
exists m : R, is_upper_bound E m.`

Definition `is_lub (E:R → Prop) (m:R) :=
is_upper_bound E m /\
(forall b:R, is_upper_bound E b → m <= b).`

Axiom `completeness : forall E:R → Prop,
bound E → (exists x : R, E x)
→ { m:R | is_lub E m }.`

Axioms of the standard library \mathbb{R}

After 17 axioms, we have defined the real field and its operations.

\Rightarrow There is left to define a bunch of lemmas to ease the proofs.

Axioms of the standard library \mathbb{R}

After 17 axioms, we have defined the real field and its operations.

\Rightarrow There is left to define a bunch of lemmas to ease the proofs.

Note this choice: Rinv is a total function. We may talk about $\frac{1}{0}$.

But the inverse property, that is $\frac{1}{x} \times x = 1$ (Rinv_l) requires that $x \neq 0$.

Hence, we cannot prove $\frac{0}{0} = 1 \dots$

Axioms of the standard library \mathbb{R}

After 17 axioms, we have defined the real field and its operations.

\Rightarrow There is left to define a bunch of lemmas to ease the proofs.

Note this choice: `Rinv` is a total function. We may talk about $\frac{1}{0}$.
But the inverse property, that is $\frac{1}{x} \times x = 1$ (`Rinv_l`) requires that $x \neq 0$.
Hence, we cannot prove $\frac{0}{0} = 1$... because it equals 0.

Another choice is that `Rinv` takes a real number and a proof of its non zeroness, but it could become cumbersome to handle proof terms inside large terms.

Automatizations about equalities

In rings, we have the ring tactic that solves equalities on rings such as \mathbb{Z} or \mathbb{R} . We create a tactic field that solves equalities on fields (including inverse). Here is the algorithm for field :

- Transform $x - y$ into $x + (-y)$ and x/y into $x \times 1/y$.
- Look for all the inverses that appear to make a product of them.
- Distribute entirely and everywhere, except in the inverses.
- Right-associate each monomial, except in the inverses.
- Multiply each side by the product of the inverses, that was constructed before, while generating the condition that it must be nonzero.
- Distribute only the product on the sum of monomials to the right and to the left without right-reassociating.
- Get rid of the inverses by using the field rule $x \times 1/x = 1$ if $x \neq 0$ and by permuting inside the monomial if needed, meaning if there are inverses left and if the field rule may apply.
- Start all over if there are inverses left.
- Call the ring tactic.

Automatizations about equalities – example (1/5)

$$x \times \left(\frac{1}{x} + \frac{x}{x+y} \right) = \left(-\frac{1}{y} \right) \times y \times \left(-\left(\frac{x \times x}{x+y} \right) - 1 \right)$$

Automatizations about equalities – example (1/5)

$$x \times \left(\frac{1}{x} + \frac{x}{x+y} \right) = \left(-\frac{1}{y} \right) \times y \times \left(- \left(\frac{x \times x}{x+y} \right) - 1 \right)$$

Transform $x - y$ into $x + (-y)$ and x/y into $x \times 1/y$.

$$x \times \left(\frac{1}{x} + x \times \frac{1}{x+y} \right) = \left(-\frac{1}{y} \right) \times y \times \left(- \left((x \times x) \times \frac{1}{x+y} \right) + (-1) \right)$$

Construct the product of inverses $p = x \times ((x+y) \times (y \times (x+y)))$.

Automatizations about equalities – example (2/5)

Distribute entirely and everywhere, except in the inverses.

$$\begin{aligned}x \times \frac{1}{x} + x \times x \times \frac{1}{x+y} &= (-1) \times \frac{1}{y} \times y \times \left((-1) \times \left((x \times x) \times \frac{1}{x+y} \right) \right) \\ &\quad + (-1) \times \frac{1}{y} \times y \times (-1)\end{aligned}$$

Right-associate each monomial, except in the inverses.

$$\begin{aligned}x \times \frac{1}{x} + x \times \left(x \times \frac{1}{x+y} \right) &= (-1) \times \left(\frac{1}{y} \times \left(y \times \left((-1) \times \left((x \times x) \times \frac{1}{x+y} \right) \right) \right) \right) \\ &\quad + (-1) \times \left(\frac{1}{y} \times (y \times (-1)) \right)\end{aligned}$$

Automatizations about equalities – example (3/5)

Multiply each side by the product of the inverses, while generating the correctness condition.

$$\begin{aligned} & (x \times ((x + y) \times (y \times (x + y)))) \times \left(x \times \frac{1}{x} + x \times \left(x \times \frac{1}{x + y} \right) \right) \\ &= (x \times ((x + y) \times (y \times (x + y)))) \times \\ & \quad \left((-1) \times \left(\frac{1}{y} \times \left(y \times \left((-1) \times \left((x \times x) \times \frac{1}{x + y} \right) \right) \right) \right) \right) \\ & \quad + (-1) \times \left(\frac{1}{y} \times (y \times (-1)) \right) \end{aligned}$$

with an additional goal $x \times ((x + y) \times (y \times (x + y))) \neq 0$.

Automatizations about equalities – example (4/5)

Distribute only the product on the sum of monomials to the right and to the left without right-reassociating.

$$\begin{aligned} & (x \times ((x + y) \times (y \times (x + y)))) \times \left(x \times \frac{1}{x}\right) \\ & + (x \times ((x + y) \times (y \times (x + y)))) \times \left(x \times \left(x \times \frac{1}{x + y}\right)\right) \\ & = (x \times ((x + y) \times (y \times (x + y)))) \\ & \quad \times \left((-1) \times \left(\frac{1}{y} \times \left(y \times \left((-1) \times \left((x \times x) \times \frac{1}{x + y}\right)\right)\right)\right)\right) \\ & + (x \times ((x + y) \times (y \times (x + y)))) \times \left((-1) \times \left(\frac{1}{y} \times (y \times (-1))\right)\right) \end{aligned}$$

Automatizations about equalities – example (5/5)

Get rid of the inverses by permuting inside the monomial if needed.

$$\begin{aligned} & ((x + y) \times (y \times ((x + y)))) \times x + (x \times (y \times (x + y))) \times x \times x = \\ & (x \times (x + y)) \times ((-1) \times (y \times ((-1) \times (x \times x)))) \\ & + (x \times ((x + y) \times (x + y))) \times ((-1) \times (y \times (-1))) \end{aligned}$$

There is no inverse left, but an equality with additions and multiplication that **ring solves**.

Other definitions in the standard library

- **limit**: for example, the function f has a limit l in x_0 is defined by

$$\forall \varepsilon > 0, \exists \alpha > 0 \text{ such that } \forall x, |x - x_0| < \alpha \rightarrow |f(x) - l| < \varepsilon$$

Other definitions in the standard library

- **limit**: for example, the function f has a limit l in x_0 is defined by

$$\forall \varepsilon > 0, \exists \alpha > 0 \text{ such that } \forall x, |x - x_0| < \alpha \rightarrow |f(x) - l| < \varepsilon$$

- **continuity, derivability** using the limit definition

Other definitions in the standard library

- **limit**: for example, the function f has a limit l in x_0 is defined by

$$\forall \varepsilon > 0, \exists \alpha > 0 \text{ such that } \forall x, |x - x_0| < \alpha \rightarrow |f(x) - l| < \varepsilon$$

- **continuity, derivability** using the limit definition
- **elementary functions** by infinite sums: \exp , \cos , \sin , \tan ...

Other definitions in the standard library

- **limit**: for example, the function f has a limit l in x_0 is defined by

$$\forall \varepsilon > 0, \exists \alpha > 0 \text{ such that } \forall x, |x - x_0| < \alpha \rightarrow |f(x) - l| < \varepsilon$$

- **continuity, derivability** using the limit definition
- **elementary functions** by infinite sums: \exp , \cos , \sin , \tan ...
- and many results about all that!

Outline

- 1 \mathbb{R} : formalization
- 2 Real-life \mathbb{R} : floating-point numbers
- 3 \mathbb{F} : formalization
- 4 Proofs about \mathbb{F}

Floating-point numbers

The memory of the computers is limited, so we limit the **precision** (number of digits) we use. Values that can be represented are called **floating-point numbers**.

$$\pi \mapsto 3.14$$

Floating-point numbers

The memory of the computers is limited, so we limit the **precision** (number of digits) we use. Values that can be represented are called **floating-point numbers**.

$$\pi \quad \hookrightarrow \quad 3.14$$

$$123456 \quad \hookrightarrow \quad 1.23 \times 10^5$$

$$\frac{1}{17} = 0.0588235\dots \quad \hookrightarrow \quad 5.88 \times 10^{-2}$$

Floating-point numbers

The memory of the computers is limited, so we limit the **precision** (number of digits) we use. Values that can be represented are called **floating-point numbers**.

$$\pi \quad \hookrightarrow \quad 3.14$$

$$123456 \quad \hookrightarrow \quad 1.23 \times 10^5$$

$$\frac{1}{17} = 0.0588235\dots \quad \hookrightarrow \quad 5.88 \times 10^{-2}$$

These examples use radix-10 and 3 digits, but the processor (FPU) uses radix-2 and 24 or 53 digits.

Floating-point numbers

They are only a sequence of bits:

11100011010010011110000111000000

Floating-point numbers

They are only a sequence of bits:

11100011010010011110000111000000

We give it a meaning depending on given sizes for s (sign), e (exponent) and f (fraction)

1	11000110	10010011110000111000000
1	11000110	10010011110000111000000
s	e	f

Floating-point numbers

and a **real value**

$$\begin{array}{ccc} \boxed{1} & \boxed{11000110} & \boxed{10010011110000111000000} \\ s & e & f \\ \downarrow & \downarrow & \downarrow \\ (-1)^s & \times 2^{e-B} & \times 1 \bullet f \\ (-1)^1 & \times 2^{198-127} & \times 1.10010011110000111000000_2 \\ & & -2^{54} \times 206727 \approx -3,7 \times 10^{21} \end{array}$$

Floating-point numbers

and a **real value**

$$\begin{array}{ccc} \boxed{1} & \boxed{11000110} & \boxed{100100111110000111000000} \\ s & e & f \\ \downarrow & \downarrow & \downarrow \\ (-1)^s & \times 2^{e-B} & \times 1 \bullet f \\ (-1)^1 & \times 2^{198-127} & \times 1.100100111110000111000000_2 \\ & -2^{54} \times 206727 \approx -3,7 \times 10^{21} & \end{array}$$

except for **special value** for e : ± 0 , $\pm \infty$, NaN, subnormals.

Floating-point numbers: subnormals

$$\begin{array}{ccc} \boxed{0} & \boxed{00000000} & \boxed{000100111110000101010000} \\ s & e & f \\ \downarrow & \downarrow & \downarrow \\ (-1)^s & \times 2^{1-B} & \times 0 \bullet f \\ \\ (-1)^0 & \times 2^{-126} & \times 0.000100111110000101010000_2 \\ & 2^{-145} \times 40469 & \approx 9.1 \times 10^{-40} \end{array}$$

± 0 can be seen as a subnormal.

Floating-point numbers: ulp

Ulp (unit in the last place) is the value of the last bit of the mantissa of a floating-point number:

- if f is a normal number $f = 1.d_{-1}d_{-2}\cdots d_{1-p} \times 2^e$, then
$$\text{ulp}(f) = 2^{e+1-p},$$
- if f is a subnormal number, $f = 1.d_{-1}d_{-2}\cdots d_{1-p} \times 2^{1-B}$ then
$$\text{ulp}(f) = 2^{2-B-p}$$

Floating-point computations

Each elementary computation is then as good as possible: the processor gives the best possible answer (IEEE-754 standard), given its requirements.

$$3.14^2 = 9.8596 \quad \hookrightarrow \quad 9.86$$

Each computation result is **rounded**.

Floating-point computations

Each elementary computation is then as good as possible: the processor gives the best possible answer (IEEE-754 standard), given its requirements.

$$3.14^2 = 9.8596 \quad \hookrightarrow \quad 9.86$$

Each computation result is **rounded**.

Here, I ignore complex functions (exponential, cosine. . .).

One floating-point computation

There are **several possible roundings** defined by the IEEE-754 standard:

- rounding to nearest, ties to zero (default rounding mode): choose the nearest floating-point number; when in the middle, choose the one having the even mantissa.
- rounding towards $+\infty$ (up)
- rounding towards $-\infty$ (down)
- rounding towards 0
- rounding to nearest, ties away from zero: choose the nearest floating-point number; when in the middle, choose the one having the biggest absolute value (used by banks).

Error of one floating-point computation

As one rounding is always the best possible, we may bound the error of one operation rounded to nearest in a format with radix β , with p digits of precision and a smallest positive floating-point number equal to $\beta^{e_{min}}$

Error of one floating-point computation

As one rounding is always the best possible, we may bound the error of one operation rounded to nearest in a format with radix β , with p digits of precision and a smallest positive floating-point number equal to $\beta^{e_{min}}$

$$|\circ(x) - x| \leq \frac{\text{ulp}(x)}{2}$$

Error of one floating-point computation

As one rounding is always the best possible, we may bound the error of one operation rounded to nearest in a format with radix β , with p digits of precision and a smallest positive floating-point number equal to $\beta^{e_{min}}$

$$|\circ(x) - x| \leq \frac{\text{ulp}(x)}{2}$$

For **normal floating-point numbers**, we have

$$|\circ(x) - x| \leq \frac{\beta^{1-p}}{2} |x|$$

For **subnormal floating-point numbers**, we have

$$|\circ(x) - x| \leq \frac{\beta^{e_{min}}}{2}$$

Which means $|\circ(x) - x| \leq 2^{-53} |x|$ or $|\circ(x) - x| \leq 2^{-1075}$ in IEEE double.

More floating-point computations

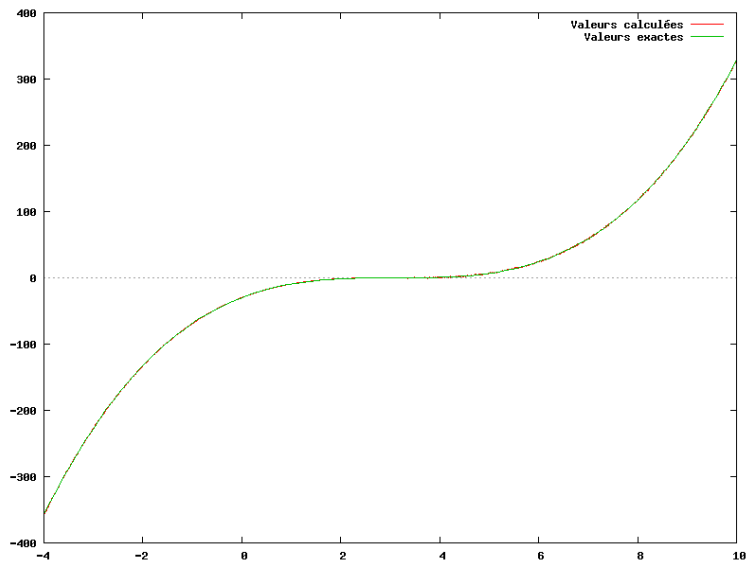
Even if one computation is nearly exact, a **sequence of computation** may be wrong:

$$\pi^2 \mapsto 3.14^2 \mapsto 9.86$$

But $\pi^2 = 9.869604\dots$ so the nearest floating-point number is 9.87.

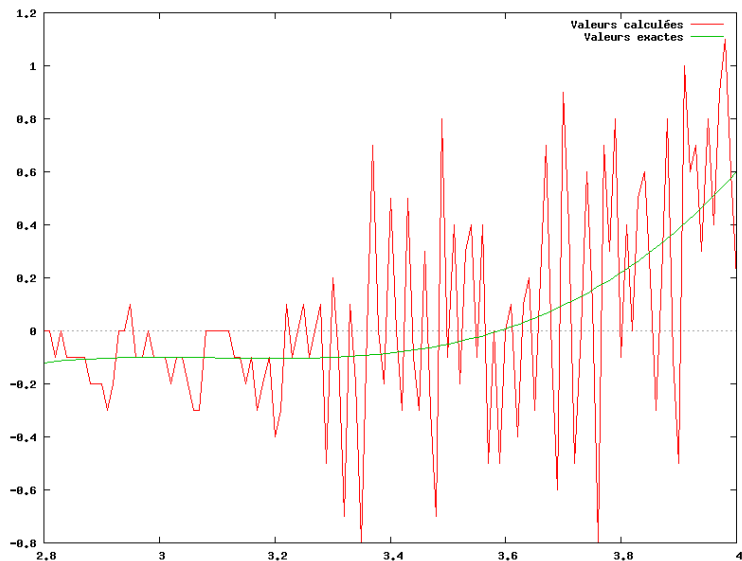
Much more floating-point computations

Let $P(x) = x^3 - 9.3x^2 + 28.8x - 29.8$.



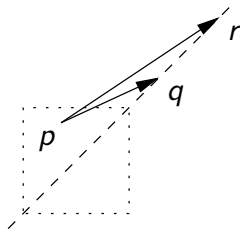
Much more floating-point computations

Let $P(x) = x^3 - 9.3x^2 + 28.8x - 29.8$.



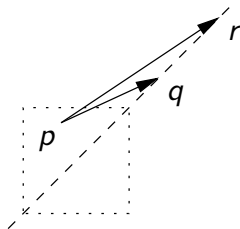
Orientation of 3 points

Given 3 points of the plane p , q and r . We want to know if pqr are right-handed ("positively" oriented) or left-handed.



Orientation of 3 points

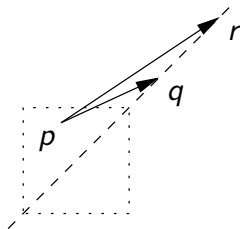
Given 3 points of the plane p , q and r . We want to know if pqr are right-handed ("positively" oriented) or left-handed.



$$\text{orient}_2(p, q, r) = \text{sign} \begin{vmatrix} q_x - p_x & r_x - p_x \\ q_y - p_y & r_y - p_y \end{vmatrix}$$

Orientation of 3 points

Given 3 points of the plane p , q and r . We want to know if pqr are right-handed ("positively" oriented) or left-handed.

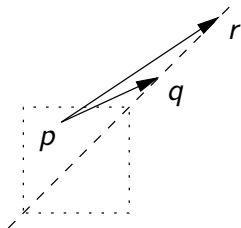


$$\text{orient}_2(p, q, r) = \text{sign} \begin{vmatrix} q_x - p_x & r_x - p_x \\ q_y - p_y & r_y - p_y \end{vmatrix}$$

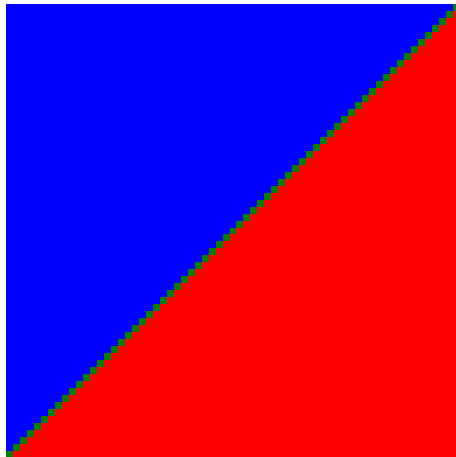
```
float det = (qx - px) * (ry - py)
           - (qy - py) * (rx - px);
if (det > 0) return POSITIVE;
if (det < 0) return NEGATIVE;
return ZERO;
```

Orientation of 3 points - exact computations

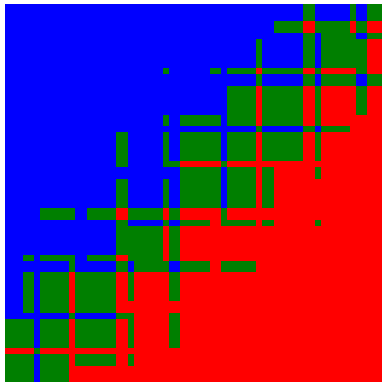
For $q = (8.1, 8.1)$ and $r = (12.1, 12.1)$ and p about $(1.5; 1.5)$,
the discriminant sign should be:



-  align 
-  orient  
-  orient  



Orientation of 3 points - single precision floating-point computations



Gulf War (1991)

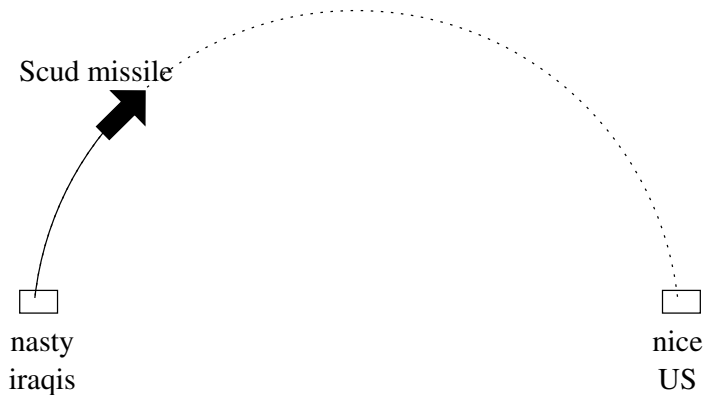


nasty
iraqis

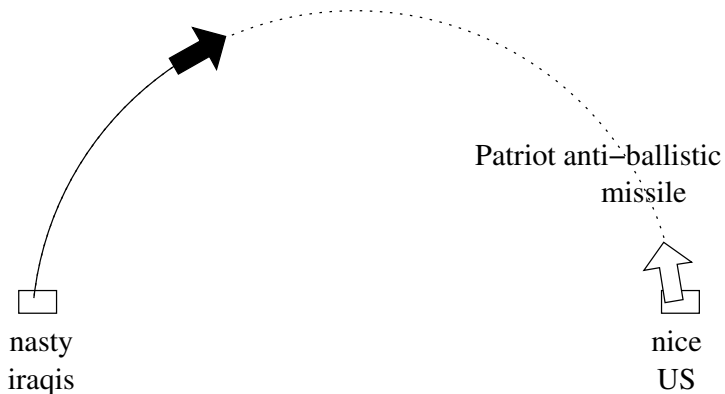


nice
US

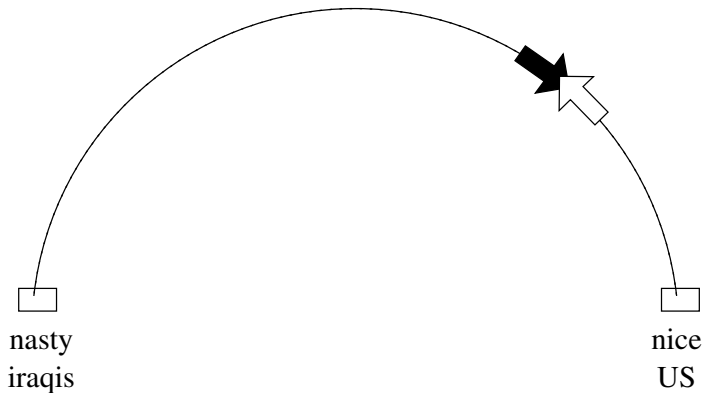
Gulf War (1991)



Gulf War (1991)



Gulf War (1991)



Gulf War (1991)



nasty
iraqis



nice
US

Gulf War - 100h later

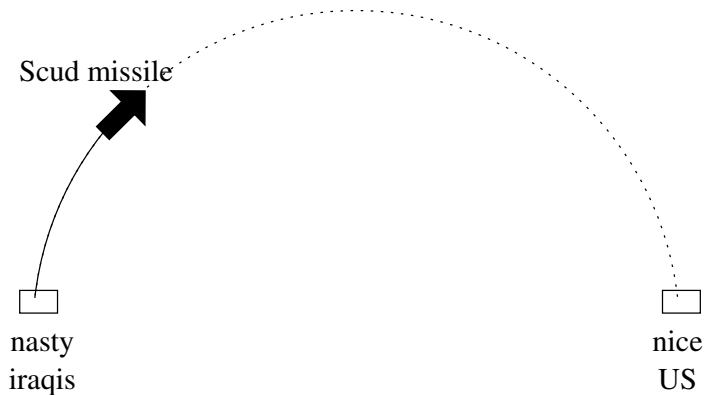


nasty
iraqis

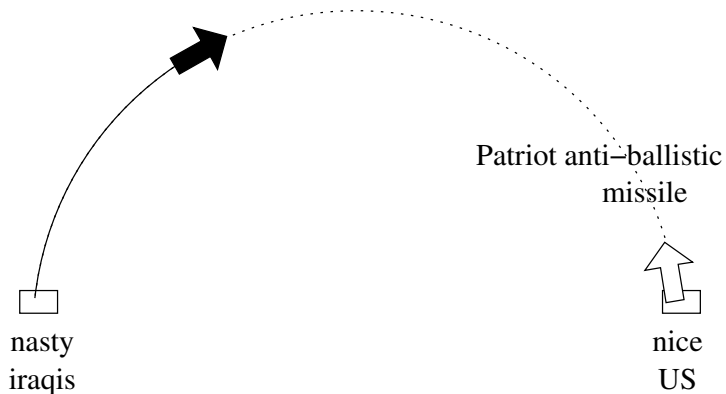


nice
US

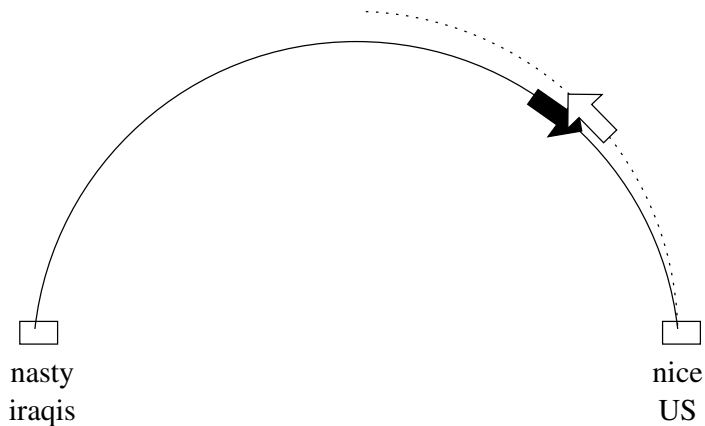
Gulf War - 100h later



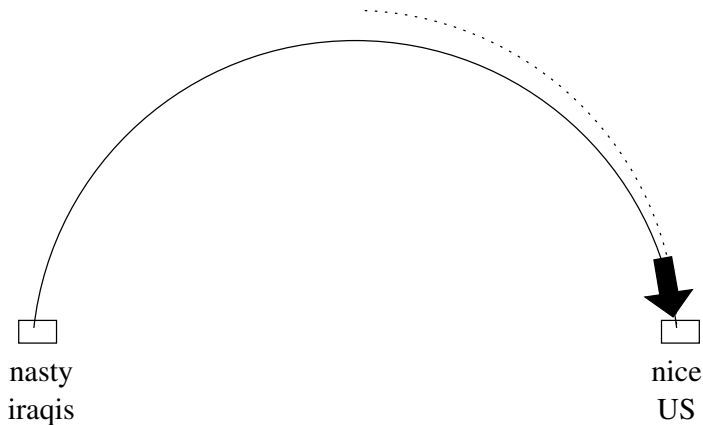
Gulf War - 100h later



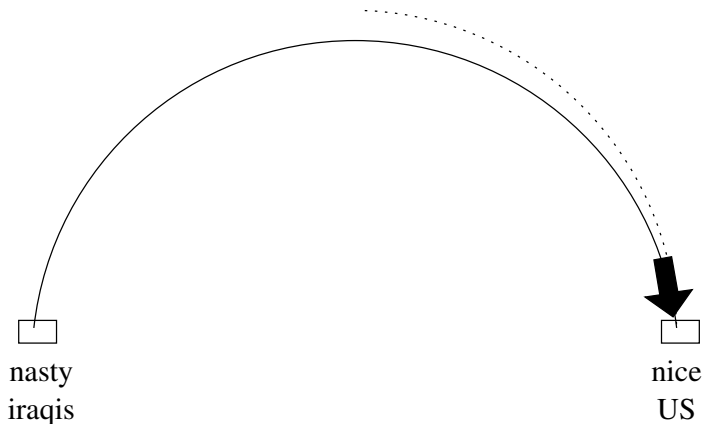
Gulf War - 100h later



Gulf War - 100h later



Gulf War - 100h later



⇒ 28 dead GI and 98 injured

Gulf War - explanation

The Patriot anti-ballistic missile is expected to work during a few hours.
But it has been working so well they let it plugged.

Gulf War - explanation

The Patriot anti-ballistic missile is expected to work during a few hours.
But it has been working so well they let it plugged.

The internal clock added 0.1s at each tick.

But 0.1 is not exact in binary:

⇒ a small rounding error at each tick

⇒ the errors, all in the same direction, accumulated

⇒ 100h later, the error was big enough to miss the missile

How can we bound a final rounding error?

- **propagate** the rounding error bounds we have for one operation (standard model)
⇒ not always useful

How can we bound a final rounding error?

- **propagate** the rounding error bounds we have for one operation (standard model)
⇒ not always useful
- **interval arithmetic**: keep an upper bound and a lower bound of your correct result, by using directed roundings
⇒ a final correct interval of unknown width

How can we bound a final rounding error?

- **propagate** the rounding error bounds we have for one operation (standard model)
⇒ not always useful
- **interval arithmetic**: keep an upper bound and a lower bound of your correct result, by using directed roundings
⇒ a final correct interval of unknown width
- use **abstract interpretation** with smart domains to handle (part of) the error compensations

How can we bound a final rounding error?

- **propagate** the rounding error bounds we have for one operation (standard model)
⇒ not always useful
- **interval arithmetic**: keep an upper bound and a lower bound of your correct result, by using directed roundings
⇒ a final correct interval of unknown width
- use **abstract interpretation** with smart domains to handle (part of) the error compensations
- **I wish I knew...**

Towards infinity and beyond!

Moreover, the computer has limits when values are too small ($\approx 10^{-300}$) or too big ($\approx 10^{300}$).

Beyond these values, the processor gives 0 or $\pm\infty$.

This means natural mathematical properties! $1 + \infty = +\infty$ and

$$-3 * +\infty = -\infty$$

Towards infinity and beyond!

Moreover, the computer has limits when values are too small ($\approx 10^{-300}$) or too big ($\approx 10^{300}$).

Beyond these values, the processor gives 0 or $\pm\infty$.

This means natural mathematical properties! $1 + \infty = +\infty$ and $-3 * +\infty = -\infty$

But

$$(10^{100})^2 \mapsto +\infty$$

Towards infinity and beyond!

Moreover, the computer has limits when values are too small ($\approx 10^{-300}$) or too big ($\approx 10^{300}$).

Beyond these values, the processor gives 0 or $\pm\infty$.

This means natural mathematical properties! $1 + \infty = +\infty$ and $-3 * +\infty = -\infty$

But

$$\begin{aligned} (10^{100})^2 &\hookrightarrow +\infty \\ \frac{1}{(10^{100})^2} &\hookrightarrow 0 \end{aligned}$$

Towards infinity and beyond!

Moreover, the computer has limits when values are too small ($\approx 10^{-300}$) or too big ($\approx 10^{300}$).

Beyond these values, the processor gives 0 or $\pm\infty$.

This means natural mathematical properties! $1 + \infty = +\infty$ and $-3 * +\infty = -\infty$

But

$$(10^{100})^2 \hookrightarrow +\infty$$

$$\frac{1}{(10^{100})^2} \hookrightarrow 0$$

$$\frac{1}{\frac{1}{(10^{100})^2}}$$

\hookrightarrow **BOOM**

Towards infinity and beyond!

Moreover, the computer has limits when values are too small ($\approx 10^{-300}$) or too big ($\approx 10^{300}$).

Beyond these values, the processor gives 0 or $\pm\infty$.

This means natural mathematical properties! $1 + \infty = +\infty$ and $-3 * +\infty = -\infty$

But

$$(10^{100})^2 \hookrightarrow +\infty$$

$$\frac{1}{(10^{100})^2} \hookrightarrow 0$$

$$\frac{1}{\frac{1}{(10^{100})^2}}$$

\hookrightarrow **BOOM**

(in fact $\hookrightarrow +\infty$ but the program stops with “division by zero”).

NaN!

By the way, the processor always gives a result.
What is the answer when asking $\sqrt{-1}$ or $+\infty - \infty$?

NaN!

By the way, the processor always gives a result.
What is the answer when asking $\sqrt{-1}$ or $+\infty - \infty$?

NaN

NaN!

By the way, the processor always gives a result.
What is the answer when asking $\sqrt{-1}$ or $+\infty - \infty$?

NaN

Not-a-Number

NaN!

NaN appears in weird places:

NaN!

NaN appears in weird places:



NaN!

NaN appears in weird places:



NaN!

NaN appears in weird places:



Bloquer

Jean Giraud, alias Moëbius, est intervenu au Festival d'Angoulême pour présenter "La Citadelle du Vertige", une nouvelle attraction du Futuroscope qui plonge le visiteur au cœur même de l'œuvre du dessinateur.

Réalisation : Bedeo.fr

Interview de Jean Giraud alias Moëbius

NaN:NaN

Source : Bédéo/Le Monde.fr

Recommandez Envoyez par email Citez Classez cet élément

NaN!

NaN appears in weird places:



Bloquer

Jean Giraud, alias Moëbius, est intervenu au Festival d'Angoulême pour présenter "La Citadelle du Vertige", une nouvelle attraction du Futuroscope qui plonge le visiteur au cœur même de l'œuvre du dessinateur.

Réalisation : Bedeo.fr

Interview de Jean Giraud alias Moëbius

NaN:NaN

Source : Bédéo/Le Monde.fr

Recommandez Envoyez par email Citez Classez cet élément

Outline

- 1 \mathbb{R} : formalization
- 2 Real-life \mathbb{R} : floating-point numbers
- 3 \mathbb{F} : formalization
- 4 Proofs about \mathbb{F}

How to formalize floating-point numbers and computations?

- what is a floating-point number? (a set, a subset of \mathbb{R} ?)
- what is a rounding?

How to formalize floating-point numbers and computations?

- what is a floating-point number? (a set, a subset of \mathbb{R} ?)
- what is a rounding?
 - ▶ a function from real numbers to floating-point numbers

How to formalize floating-point numbers and computations?

- what is a floating-point number? (a set, a subset of \mathbb{R} ?)
- what is a rounding?
 - ▶ a function from real numbers to floating-point numbers
 - ▶ a property between a real number and a floating-point number

How to formalize floating-point numbers and computations?

- what is a floating-point number? (a set, a subset of \mathbb{R} ?)
- what is a rounding?
 - ▶ a function from real numbers to floating-point numbers
“calculable”
 - ▶ a property between a real number and a floating-point number
may express roundings to nearest

Coq formalization (by L. Théry, L. Rideau, M. Daumas)

Float = pair of signed integers (mantissa, exponent)

$$(n, e) \in \mathbb{Z}^2$$

Coq formalization (by L. Théry, L. Rideau, M. Daumas)

Float = pair of signed integers (mantissa, exponent)
associated to a real value.

$$(n, e) \in \mathbb{Z}^2 \mapsto n \times \beta^e \in \mathbb{R}$$

Coq formalization (by L. Théry, L. Rideau, M. Daumas)

Float = pair of signed integers (mantissa, exponent)
associated to a real value.

$$(n, e) \in \mathbb{Z}^2 \mapsto n \times \beta^e \in \mathbb{R}$$

$$\begin{array}{ccccc} 1.00010_2 \text{ E } 4 & \mapsto & (100010_2, -1)_2 & \mapsto & 17 \\ \text{IEEE-754} & & \text{significant} & & \text{real value} \end{array}$$

\Rightarrow normal floats, subnormal floats,

Floats may have the same real value, but there is a canonical representant.

Roundings are properties between a real and a floating-point number.

Coq formalization (by G. Melquiond and myself)

For a real number x , define $\text{msb}(x)$ as being the integer such that

$$\beta^{\text{msb}(x)-1} \leq |x| < \beta^{\text{msb}(x)}.$$

A format is defined by a function $\text{fexp} : \mathbb{Z} \rightarrow \mathbb{Z}$. The value x is in the format if it can be expressed with an integer n as:

$$x = n \times \beta^{\text{fexp}(\text{msb}(x))}.$$

Coq formalization (by G. Melquiond and myself)

The `fexp` function is then defined as:

- For a fixed-point format with exponent e_{min} , we set

$$\text{FIX_fexp}(e) = e_{min}.$$

Roundings are defined both by properties and functions (with a choice for roundings to nearest).

Coq formalization (by G. Melquiond and myself)

The `fexp` function is then defined as:

- For a fixed-point format with exponent e_{min} , we set

$$\text{FIX_fexp}(e) = e_{min}.$$

- For a floating-point format with p digits and no underflow, we set

$$\text{FLX_fexp}(e) = e - p.$$

Roundings are defined both by properties and functions (with a choice for roundings to nearest).

Coq formalization (by G. Melquiond and myself)

The `fexp` function is then defined as:

- For a fixed-point format with exponent e_{min} , we set

$$\text{FIX_fexp}(e) = e_{min}.$$

- For a floating-point format with p digits and no underflow, we set

$$\text{FLX_fexp}(e) = e - p.$$

- For a floating-point format with p digits and underflow at exponent e_{min} (such as IEEE-754 double precision with 53 and -1074), we set

$$\text{FLT_fexp}(e) = \max(e - p, e_{min}).$$

Roundings are defined both by properties and functions (with a choice for roundings to nearest).

Outline

- 1 \mathbb{R} : formalization
- 2 Real-life \mathbb{R} : floating-point numbers
- 3 \mathbb{F} : formalization
- 4 Proofs about \mathbb{F}

What can be proved about floating-point numbers

- formalizations are good

What can be proved about floating-point numbers

- formalizations are good
- but using them is necessary to show their usability

What can be proved about floating-point numbers

- formalizations are good
- but using them is necessary to show their usability
- hence a few examples. . .

Theorem (Sterbenz)

If x and y are FP numbers in a given precision such that

$$\frac{y}{2} \leq x \leq 2y,$$

then $x - y$ fits in a FP number in the same precision and is therefore computed without error.

Theorem (Veltkamp/Dekker)

Provided no Overflow and no Underflow occur, there is an algorithm computing the exact error of the multiplication using only FP operations.

Theorem (Veltkamp/Dekker)

Provided no Overflow and no Underflow occur, there is an algorithm computing the exact error of the multiplication using only FP operations.

Idea:
split your floats in 2, multiply all the parts, add them in the correct order.

Theorem (Veltkamp/Dekker)

Let $C = 2^{27} + 1$ and let x and y be floating-point numbers. We compute:

- $px = x * C$; $qx = x - px$; $hx = px + qx$; $tx = x - hx$; then $x = hx + tx$.
- $py = y * C$; $qy = y - py$; $hy = py + qy$; $ty = y - hy$; then $y = hy + ty$.
- $r1 = x * y$; $r2 = (((-xy + hx * hy) + hx * ty) + hy * tx) + tx * ty$; then

$$x \times y = r1 + r2$$

provided no underflow or overflow occur (this can be exactly stated).

Accurate discriminant

It is pretty hard to compute $b^2 - 4ac$ accurately.

Accurate discriminant

It is pretty hard to compute $b^2 - ac$ accurately.

Theorem (Kahan)

*Provided no Overflow and no Underflow occur, there is an algorithm computing the $b^2 - a * c$ within 2 ulps.*

Accurate discriminant

```
double discriminant(double a, double b, double c) {  
    double p,q,d,dp,dq;  
    p=b*b;  
    q=a*c;  
  
    if (p+q <= 3*fabs(p-q))  
        d=p-q;  
    else {  
        dp=Dekker(b,b,p);  
        dq=Dekker(a,c,q);  
        d=(p-q)+(dp-dq);  
    }  
    return d;  
}
```

Accurate discriminant

```
double discriminant(double a, double b, double c) {  
    double p,q,d,dp,dq;  
    p=b*b;  
    q=a*c;  
  
    if (p+q <= 3*fabs(p-q))  
        d=p-q;  
    else {  
        dp=Dekker(b,b,p);  
        dq=Dekker(a,c,q);  
        d=(p-q)+(dp-dq);  
    }  
    return d;  
}
```

Various issues: Underflow, Overflow, cancellation, incorrect test.

Many other proofs

- accurate **polynomial evaluation** (for a class of polynomials including the ones used for elementary functions evaluation)

Many other proofs

- accurate **polynomial evaluation** (for a class of polynomials including the ones used for elementary functions evaluation)
- **argument reduction** (first step of elementary functions evaluation)

Many other proofs

- accurate **polynomial evaluation** (for a class of polynomials including the ones used for elementary functions evaluation)
- **argument reduction** (first step of elementary functions evaluation)
- **floating-point tricks** for a fast computing the floating-point successor or predecessor

Many other proofs

- accurate **polynomial evaluation** (for a class of polynomials including the ones used for elementary functions evaluation)
- **argument reduction** (first step of elementary functions evaluation)
- **floating-point tricks** for a fast computing the floating-point successor or predecessor
- floating-point tricks about the **FMA** ($\circ(x \times y + z)$): exact and approximated errors

Many other proofs

- accurate **polynomial evaluation** (for a class of polynomials including the ones used for elementary functions evaluation)
- **argument reduction** (first step of elementary functions evaluation)
- **floating-point tricks** for a fast computing the floating-point successor or predecessor
- floating-point tricks about the **FMA** ($\circ(x \times y + z)$): exact and approximated errors
- **applied mathematics**: simple numerical scheme for solving a PDE

Many other proofs

- accurate **polynomial evaluation** (for a class of polynomials including the ones used for elementary functions evaluation)
- **argument reduction** (first step of elementary functions evaluation)
- **floating-point tricks** for a fast computing the floating-point successor or predecessor
- floating-point tricks about the **FMA** ($\circ(x \times y + z)$): exact and approximated errors
- **applied mathematics**: simple numerical scheme for solving a PDE
- ...

Conclusion

- Very high guarantee

Conclusion

- Very high guarantee
- All the power of Coq to prove:

Conclusion

- Very high guarantee
- All the power of Coq to prove:
 - ▶ not only rounding errors, but also

Conclusion

- Very high guarantee
- All the power of Coq to prove:
 - ▶ not only rounding errors, but also
 - ▶ link with mathematical properties

Conclusion

- Very high guarantee
- All the power of Coq to prove:
 - ▶ not only rounding errors, but also
 - ▶ link with mathematical properties
 - ▶ floating-point tricks

Conclusion: limits (1/2)

- long and tedious

Conclusion: limits (1/2)

- long and tedious \Rightarrow automations!

Conclusion: limits (1/2)

- long and tedious \Rightarrow automations!
- for example Gappa (G. Melquiond)

Conclusion: limits (1/2)

- long and tedious \Rightarrow automations!
 - for example Gappa (G. Melquiond)
- \Rightarrow Use automatic provers inside Coq to ease proofs

Conclusion: limits (2/2)

- An algorithm is not a program: memory model, overflows. . .

Conclusion: limits (2/2)

- An algorithm is not a program: memory model, overflows. . .
- A program is not a compiled program: optimizations, compilation discrepancies. . .

Perspectives

- certification of numerical analysis programs

Perspectives

- certification of numerical analysis programs
- certification of the certifiers
(abstract interpretation, static analysis tools, compilers)

Perspectives

- certification of numerical analysis programs
- certification of the certifiers
(abstract interpretation, static analysis tools, compilers)
- certification of basic libraries
(multiprecision libraries, linear algebra libraries)