Some issues related to double roundings

Erik Martin-Dorel[1] Guillaume Melquiond[2] Jean-Michel Muller[3]

[1]ENS Lyon, [2]Inria, [3]CNRS

Valencia, June 2012

Floating-Point arithmetic, very quickly...

Assuming extremal exponents $e_{\min}$ and $e_{\max}$, a finite, precision-p, binary FP number x is of the form

$$x = M \cdot 2^{e-p+1}, \tag{1}$$

M and e: integers such that

$$\begin{cases} |M| \leq 2^p - 1 \\ e_{\min} \leq e \leq e_{\max} \end{cases} \tag{2}$$

- Largest M (in magnitude) such that (1) and (2) hold: integral significand of x;
- corresponding value of e (for $x \neq 0$): exponent of x;
- subnormal number: $e = e_{\min}$ and $|M| < 2^{p-1}$ (assumed available).

IEEE 754: correctly rounded operations

Definition 1 (Correct rounding)

Rounding function $\circ$, chosen among:

- toward $-\infty$: $\mathrm{RD}(x)$ is the largest FP number $\leq x$;
- toward $+\infty$: $\mathrm{RU}(x)$ is the smallest FP number $\geq x$;
- toward zero: $\mathrm{RZ}(x)$ is equal to $\mathrm{RD}(x)$ if $x \geq 0$, and to $\mathrm{RU}(x)$ if $x \leq 0$;
- to nearest: $\mathrm{RN}(x) =$ FP number closest to x. In case of a tie: the one whose integral significand is even (another tie-breaking rule: away from 0)

Correctly rounded operation $\top$: returns $\circ(a \top b)$ for all FP numbers a and b.

IEEE 754: correctly rounded operations

IEEE 754-1985: Correct rounding for $+$, $-$, $\times$, $\div$, $\sqrt{}$ and some conversions. Advantages:

- if the result of an operation is exactly representable, we get it;
- if we just use the 4 arith. operations and $\sqrt{}$, deterministic arithmetic: $\rightarrow$ algorithms and proofs that use the specifications;
- accuracy and portability are improved;
- ...

FP arithmetic becomes a structure in itself, that can be studied.

First example: Sterbenz Lemma

Lemma 2 (Sterbenz)

Let a and b be positive FP numbers. If

$$\frac{a}{2} \leq b \leq 2a$$

then $a - b$ is a FP number ($\rightarrow$ computed exactly, in any rounding mode).

Proof: straightforward using the notation $x = M \times 2^{e+1-p}$.

Error of rounded-to-nearest FP addition

Reminder: $\text{RN}(x)$ is x rounded to nearest.

Lemma 3

Let a and b be two FP numbers. Let

$$s = \text{RN}(a + b)$$

and

$$r = (a + b) - s.$$

If no overflow when computing s, then r is a FP number.

$\rightarrow$ the error of a FP addition is exactly representable by a FPN.

Error of FP addition

Proof: Assume $|a| \geq |b|$,

1. s is "the" FP number nearest $a+b \rightarrow$ it is closest to $a+b$ than a. Hence $|(a+b)-s| \leq |(a+b)-a|$, therefore

$$|r| \leq |b|.$$

Error of FP addition

Proof: Assume $|a| \geq |b|$,

1. s is "the" FP number nearest $a+b \rightarrow$ it is closest to $a+b$ than a. Hence $|(a+b)-s| \leq |(a+b)-a|$, therefore
$$|r| \leq |b|.$$
2. denote $a = M_a \times 2^{e_a-p+1}$ and $b = M_b \times 2^{e_b-p+1}$, with $|M_a|, |M_b| \leq 2^p - 1$, M_a and M_b largest, and $e_a \geq e_b$.

Error of FP addition

Proof: Assume $|a| \geq |b|$,

1. s is "the" FP number nearest $a+b \rightarrow$ it is closest to $a+b$ than a. Hence $|(a+b)-s| \leq |(a+b)-a|$, therefore
$$|r| \leq |b|.$$
2. denote $a = M_a \times 2^{e_a-p+1}$ and $b = M_b \times 2^{e_b-p+1}$, with $|M_a|, |M_b| \leq 2^p - 1$, M_a and M_b largest, and $e_a \geq e_b$.

   $a+b$ is multiple of $2^{e_b-p+1} \Rightarrow s$ and r are multiple of 2^{e_b-p+1} too $\Rightarrow \exists R \in \mathbb{Z}$ s.t.
$$r = R \times 2^{e_b-p+1}$$

Error of FP addition

Proof: Assume $|a| \geq |b|$,

1. s is "the" FP number nearest $a+b \rightarrow$ it is closest to $a+b$ than a. Hence $|(a+b)-s| \leq |(a+b)-a|$, therefore

$$|r| \leq |b|.$$

2. denote $a = M_a \times 2^{e_a-p+1}$ and $b = M_b \times 2^{e_b-p+1}$, with $|M_a|, |M_b| \leq 2^p - 1$, M_a and M_b largest, and $e_a \geq e_b$.

   $a+b$ is multiple of $2^{e_b-p+1} \Rightarrow s$ and r are multiple of 2^{e_b-p+1} too $\Rightarrow \exists R \in \mathbb{Z}$ s.t.

$$r = R \times 2^{e_b-p+1}$$

   but, $|r| \leq |b| \Rightarrow |R| \leq |M_b| \leq 2^p - 1 \Rightarrow$ r is a FP number.

Get r: the fast2sum algorithm (Dekker)

Theorem 4 (Fast2Sum (Dekker))

Subnormal numbers available, no overflows. FP numbers a and b s.t. $e_a \geq e_b$. Following algorithm:

- $s + r = a + b$ *exactly;*
- s *is "the" FP number that is closest to* $a + b$.

Algorithm 1 (FastTwoSum)

$s \leftarrow \mathrm{RN}(a + b)$
$z \leftarrow \mathrm{RN}(s - a)$
$r \leftarrow \mathrm{RN}(b - z)$

C Program 1

```
s = a+b;
z = s-a;
r = b-z;
```

Important remark: Proving the behavior of such algorithms requires use of the correct rounding property.

The 2Sum algorithm (Knuth)

Does not require comparison of a and b.

Algorithm 2 (2Sum(a, b))

$$s \leftarrow \mathrm{RN}(a+b)$$
$$a' \leftarrow \mathrm{RN}(s-b)$$
$$b' \leftarrow \mathrm{RN}(s-a')$$
$$\delta_a \leftarrow \mathrm{RN}(a-a')$$
$$\delta_b \leftarrow \mathrm{RN}(b-b')$$
$$t \leftarrow \mathrm{RN}(\delta_a+\delta_b)$$

If a and b are normal FPN, then $a+b=s+t$.

So we do live in the best of all possible worlds. . .

- correct rounding $\longrightarrow$ "deterministic arithmetic";

So we do live in the best of all possible worlds. . .

- correct rounding $\rightarrow$ "deterministic arithmetic";
- we easily compute the error of a FP addition (by the way: same for $\times$);

So we do live in the best of all possible worlds. . .

- correct rounding $\longrightarrow$ "deterministic arithmetic";
- we easily compute the error of a FP addition (by the way: same for $\times$);
- we can re-inject that error later on in a calculation, to compute accurate sums, dot-products. . .

So we do live in the best of all possible worlds...

- correct rounding $\longrightarrow$ "deterministic arithmetic";
- we easily compute the error of a FP addition (by the way: same for $\times$);
- we can re-inject that error later on in a calculation, to compute accurate sums, dot-products...
- already many such algorithms, probably more to come.

So we do live in the best of all possible worlds...

- correct rounding $\longrightarrow$ "deterministic arithmetic";
- we easily compute the error of a FP addition (by the way: same for $\times$);
- we can re-inject that error later on in a calculation, to compute accurate sums, dot-products...
- already many such algorithms, probably more to come.

...except I'm a liar!

Deterministic arithmetic?

- several FP formats in a given environment → difficult to know in which format some operations are performed;
- may make the result of a sequence of operations difficult to predict;

Assume all declared variables are of the same format. Two phenomenons may occur when a wider format is available:

- implicit variables such as result of "`a+b`" in "`d = (a+b)*c`": not clear in which format they are computed;
- explicit variables may be first computed (hence rounded) in the wider format, and then rounded again to the destination format.

Deterministic arithmetic?

C program:

```
double a = 1848874847.0;
double b = 19954562207.0;
double c;
c = a * b;
printf("c = %20.19e\n", c);
return 0;
```

Depending on the environment, `3.6893488147419103232e+19` or `3.6893488147419111424e+19` (binary64 number closest to the exact product).

What happened?

Exact value of `a*b`: 36893488147419107329. Binary representation:

$$\overbrace{\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}\,10000000000}^{64\text{ bits}}\,01$$

If the product is first rounded to "double-extended precision", we get

$$\overbrace{\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}\,10000000000}^{64\text{ bits}} \times 4$$

if that intermediate value is rounded to the binary64 destination format,

$$\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}} \times 2^{13}$$

$= 36893488147419103232_{10}$,

$\rightarrow$ rounded down, whereas it should have been rounded up.

Is it a problem ?

- these phenomenons: almost always innocuous (error very slightly above 1/2 ulp);
- they may make the behavior of some programs difficult to predict;
- most compilers offer options that prevent this problem. And yet, needing these options
  - restricts the portability of numerical programs;
  - possible bad impact on performance and/or accuracy.

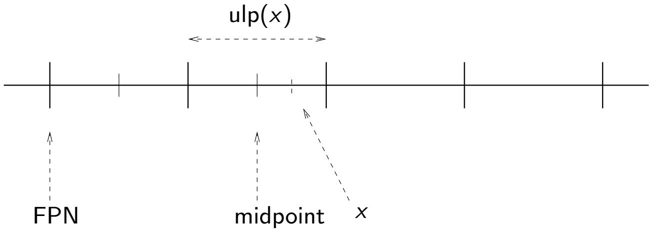
$\longrightarrow$ examine which properties remain true when double roundings occur.

Double roundings: known issues

- $\pm$, $\times$, $\div$, $\sqrt{\ }$: conditions on the precision of the wider format under which double roundings do not change the result (Kahan, Figueroa's PhD—2000);
- double roundings may cause a problem in binary to decimal conversions. Solutions given by Goldberg, and by Cornea et al;
- double roundings may occur, even without available wider format, when performing scaled division iterations to avoid overflow or underflow;
- Rounding towards $\pm\infty$ or 0: double roundings do not change the result of a calculation $\rightarrow$ we focus on "round to nearest" only.

Notation

- precision-p target format, and precision-$(p + p')$ "internal" format;
- $\mathrm{RN}_k(u)$ means u rounded to the nearest precision-k FP number;
- when the precision is omitted: it is p;
- precision-p midpoint: exactly halfway between two consecutive precision-p FPN.



Throughout the presentation: we assume that no overflow occurs.

Double roundings and double rounding slips

When the arithmetic operation $x \top y$ appears in a program:

- double rounding: what is actually performed is

$$\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(x \top y)\right),$$

- double rounding slip: a double rounding occurs and the obtained result differs from $\mathrm{RN}_p(x \top y)$.

Remark 1

Double rounding slip $\rightarrow$ the error of $a + b$ may not be a FPN.

Double rounding → the error of $a + b$ may not be a FPN

Consider $a = 1\underbrace{xxxx\cdots x}_{p-3 \text{ bits}}01$, where $xxxx\cdots x$ is any $(p-3)$-bit bit-chain.

Also consider, $b = 0.0\underbrace{111111\cdots 1}_{p \text{ ones}} = \frac{1}{2} - 2^{-p-1}$. We have:

$$a + b = \underbrace{1xxxx...x01}_{p \text{ bits}}.0\underbrace{111111...1}_{p \text{ bits}},$$

so that if $1 \le p' \le p$, $u = RN_{p+p'}(a+b) = 1xxxx...x01.100...0$,

Double rounding $\rightarrow$ the error of $a+b$ may not be a FPN

$$u = RN_{p+p'}(a+b) = 1xxxx...x01.100...0,$$

The "round to nearest even" rule thus implies

$$s = RN_p(u) = 1xxxx...x10 = a+1$$

Therefore,

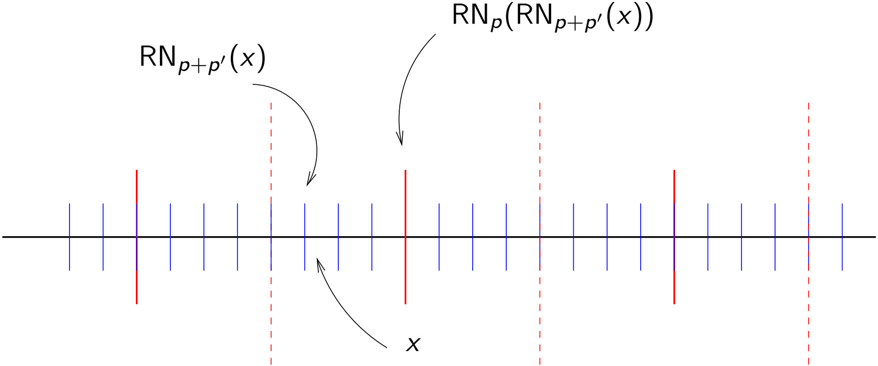
$$s-(a+b) = a+1-(a+\frac{1}{2}-2^{-p-1}) = \frac{1}{2}+2^{-p-1} = 0.\underbrace{10000\cdots01}_{p+1 \text{ bits}},$$

which is not exactly representable in precision-p FP arithmetic.

A few preliminary remarks

Remark 2

If a double rounding slip occurs when evaluating $a \top b$ then $\mathrm{RN}_{p+p'}(a \top b)$ is a precision-p midpoint, i.e., a number exactly halfway between two consecutive precision-p FP numbers.



A few preliminary remarks

Remark 3

Since the precision-p FPNs are precision-$(p+p')$ FPNs, each time $a \top b$ is exactly representable in precision-p arithmetic, we get it:

$$\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a \top b)\right) = \mathrm{RN}_p(a \top b) = a \top b.$$

$\longrightarrow$ Sterbenz Lemma still holds in presence of double roundings.

Remark 4

Let a and b be precision-p FP numbers, and define

$$s = \mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a+b)\right).$$

$a+b-s$ fits in at most $p+2$ bits, so that as soon as $p' \geq 2$, we have

$$\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a+b-s)\right) = \mathrm{RN}_p(a+b-s). \quad (3)$$

Fast2Sum and double roundings

Algorithm 3 (Fast2Sum-with-double-roundings(a, b))

$$s \leftarrow \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}(a+b)\right]$$
$$z \leftarrow \circ(s-a)$$
$$t \leftarrow \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}(b-z)\right]$$

$\circ(u)$: $\mathrm{RN}_p(u)$, $\mathrm{RN}_{p+p'}(u)$, or $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(u))$, or any faithful rounding.

Fast2Sum and double roundings

Theorem 5

If $p \geq 3$, $p' \geq 2$, and a and b are precision-p FPN with $e_a \geq e_b$, then Algorithm 3 satisfies:

- *$z = s - a$ exactly;*
- *if no double rounding slip occurred when computing s (i.e., if $s = \mathrm{RN}_p(a + b)$), then $t = (a + b - s)$ exactly;*
- *otherwise, $t = \mathrm{RN}_p(a + b - s)$.*

The proof of Theorem 5 is rather complex (many sub-cases). We have a formal proof that uses the Coq proof assistant.

Proof in the (much simpler) case $|a| \geq |b|$

1. if a and b have same sign, $|a| \leq |a+b| \leq |2a|$, hence ($2a$ is a FPN, rounding is increasing) $|a| \leq |s| \leq |2a| \rightarrow$ (Sterbenz) $z = s - a$. Therefore $t = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}(b-z)\right] = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}((a+b)-s)\right]$.

Proof in the (much simpler) case $|a| \geq |b|$

1. if a and b have same sign, $|a| \leq |a+b| \leq |2a|$, hence ($2a$ is a FPN, rounding is increasing) $|a| \leq |s| \leq |2a| \rightarrow$ (Sterbenz) $z = s - a$. Therefore $t = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}(b-z)\right] = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}((a+b)-s)\right]$.
2. if a and b have opposite signs then
   - either $|b| \geq |a/2|$, which implies (Sterbenz) $a+b$ is a FPN, thus $s = a+b$, $z = b$ and $t = 0$;
   - or $|b| < |a/2|$, which implies $|a+b| > |a/2|$, hence $s \geq |a/2|$, thus (Sterbenz) $z = s - a$. Therefore $t = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}(b-z)\right] = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}((a+b)-s)\right]$.

Proof in the (much simpler) case $|a| \geq |b|$

1. if a and b have same sign, $|a| \leq |a+b| \leq |2a|$, hence ($2a$ is a FPN, rounding is increasing) $|a| \leq |s| \leq |2a| \rightarrow$ (Sterbenz) $z = s - a$. Therefore $t = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}(b-z)\right] = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}((a+b)-s)\right]$.
2. if a and b have opposite signs then
   - either $|b| \geq |a/2|$, which implies (Sterbenz) $a+b$ is a FPN, thus $s = a+b$, $z = b$ and $t = 0$;
   - or $|b| < |a/2|$, which implies $|a+b| > |a/2|$, hence $s \geq |a/2|$, thus (Sterbenz) $z = s - a$. Therefore $t = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}(b-z)\right] = \mathrm{RN}_p\left[\mathrm{RN}_{p+p'}((a+b)-s)\right]$.

Remark 4 $\Rightarrow t = \mathrm{RN}_p((a+b)-s)$.

2Sum and double roundings

Algorithm 4 (2Sum-with-double-roundings(a, b))

(1) $s \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a+b))$ *or* $\mathrm{RN}_p(a+b)$
(2) $a' \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(s-b))$ *or* $\mathrm{RN}_p(s-b))$
(3) $b' \leftarrow \circ(s-a')$
(4) $\delta_a \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a-a'))$ *or* $\mathrm{RN}_p(a-a')$
(5) $\delta_b \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(b-b'))$ *or* $\mathrm{RN}_p(b-b')$
(6) $t \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(\delta_a+\delta_b))$ *or* $\mathrm{RN}_p(\delta_a+\delta_b)$

$\circ(u)$: $\mathrm{RN}_p(u)$, $\mathrm{RN}_{p+p'}(u)$, or $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(u))$, or any faithful rounding.

Theorem 6

If $p \geq 4$ and $p + p'$, with $p' \geq 2$. If a and b are precision-p FPN, and if no overflow occurs, then Algorithm 4 satisfies:

- *if no double rounding slip occurred when computing s then $t = (a + b - s)$ exactly;*
- *otherwise, $t = \mathrm{RN}_p(a + b - s)$.*

Proofs and tech. report available at
`http://hal-ens-lyon.archives-ouvertes.fr/ensl-00644408`
(submitted to a journal)

u and γ_k notations

- Higham's notations, very slightly adapted to the context of double roundings.
- Define $u = 2^{-p}$ and $u' = 2^{-p} + 2^{-p-p'} + 2^{-2p-p'}$. For any integer $k \ll 2^p$, define

$$\gamma_k = \frac{ku}{1 - ku} \approx k \cdot 2^{-p},$$

- and

$$\gamma'_k = \frac{ku'}{1 - ku'} \approx k \cdot (2^{-p} + 2^{-p-p'}).$$

Application: summation algorithms

Naive, recursive-sum algorithm, rewritten with double roundings.

Algorithm 5

$r \leftarrow a_1$
for $i = 2$ *to* n ***do***
  $r \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(r + a_i))$
end for
return r

Property 1

$$\left| r - \sum_{i=1}^{n} a_i \right| \leq \gamma'_{n-1} \sum_{i=1}^{n} |a_i|.$$

Without double roundings, the bound is $\gamma_{n-1} \sum_{i=1}^{n} |a_i|$.

Rump, Ogita and Oishi's K-fold summation algorithm

Algorithm 6 (VecSum(a), where $a = (a_1, a_2, \ldots, a_n)$)

$p \leftarrow a$
for $i = 2$ *to* n ***do***
  $(p_i, p_{i-1}) \leftarrow \mathit{2Sum}(p_i, p_{i-1})$
end for
return p

Algorithm 7 (K-fold summation algorithm)

for $k = 1$ *to* $K - 1$ ***do***
  $a \leftarrow \mathit{VecSum}(a)$
end for
$c = a_1$
for $i = 2$ *to* $n - 1$ ***do***
  $c \leftarrow \mathrm{RN}(c + a_i)$
end for
return $\mathrm{RN}(a_n + c)$

Rump, Ogita and Oishi's K-fold summation algorithm

- without double roundings, if $4nu < 1$, the FPN σ returned by Algorithm 7 satisfies

$$\left|\sigma - \sum_{i=1}^{n} a_i\right| \leq (u + \gamma_{n-1}^2)\left|\sum_{i=1}^{n} a_i\right| + \gamma_{2n-2}^{K} \sum_{i=1}^{n} |a_i|. \qquad (4)$$

- if a double-rounding slip occurs in the first call to VecSum, not possible to show an error bound better than prop. to $2^{-2p} \sum_{i=1}^{n} |a_i|$;

Rump, Ogita and Oishi's K-fold summation algorithm

Example (with $n = 5$, but easily generalizable):

$$(a_1, a_2, a_3, a_4, a_5) = \left(2^{p-1} + 1\ ,\ \frac{1}{2} - 2^{-p-1}\ ,\ -2^{p-1}\ ,\ -2\ ,\ \frac{1}{2}\right)$$

- Algorithm 7 run with double roundings, with $1 \leq p' \leq p$;
- in the first addition $(a_1 + a_2)$, double rounding slip $\rightarrow$ after the first Fast2Sum, $p_2 = 2^{p-1} + 2$ and $p_1 = -1/2$, so that $p_1 + p_2 \neq a_1 + a_2$;
- At the end of the first call to VecSum, the returned vector is

$$\left(-\frac{1}{2}, 0, 0, 0, \frac{1}{2}\right)$$

Rump, Ogita and Oishi's K-fold summation algorithm

Example (with $n = 5$, but easily generalizable):

$$(a_1, a_2, a_3, a_4, a_5) = \left(2^{p-1} + 1\ ,\ \frac{1}{2} - 2^{-p-1}\ ,\ -2^{p-1}\ ,\ -2\ ,\ \frac{1}{2}\right)$$

- Algorithm 7 run with double roundings, with $1 \leq p' \leq p$;
- in the first addition $(a_1 + a_2)$, double rounding slip → after the first Fast2Sum, $p_2 = 2^{p-1} + 2$ and $p_1 = -1/2$, so that $p_1 + p_2 \neq a_1 + a_2$;
- At the end of the first call to VecSum, the returned vector is

$$\left(-\frac{1}{2}, 0, 0, 0, \frac{1}{2}\right)$$

→ Algorithm 7 returns 0, $\forall K$, whereas $\sum a_i = -2^{-p-1}$. Final error $\approx 2^{-2p-1} \sum |a_i|$, $\forall K$.

Conclusion

- investigated possible influence of double roundings on several algorithms of the FP literature;
- many important properties are preserved;
- depending on the considered applications, these properties may suffice, or specific compilation options should be chosen to prevent double roundings;
- hopefully, implementation of IEEE 754-2008 will bring some help;
- some proofs (e.g., 2Sum) long and tricky $\rightarrow$ formal proof.