Some issues related to double roundings

Érik Martin-Dorel, Guillaume Melquiond, Jean-Michel Muller

Noname manuscript No.
(will be inserted by the editor)

Some issues related to double rounding

Érik Martin-Dorel · Guillaume Melquiond · Jean-Michel Muller

Received: date / Accepted: date

Abstract Double rounding is a phenomenon that may occur when different floating-point precisions are available on the same system. Although double rounding is, in general, innocuous, it may change the behavior of some useful small floating-point algorithms. We analyze the potential influence of double rounding on the Fast2Sum and 2Sum algorithms, on some summation algorithms, and Veltkamp's splitting.

Keywords Floating-point arithmetic · Double rounding · Correct rounding · 2Sum · Fast2Sum · Summation algorithms

Mathematics Subject Classification (2000) 65G99 · 65Y04 · 68M15

1 Introduction

When several floating-point formats are supported in a given environment, it is sometimes difficult to know in which format some operations are performed. This may make the result of a sequence of arithmetic operations difficult to predict, unless adequate compilation switches are selected. This is an issue addressed by the recent IEEE 754-2008 standard for floating-point arithmetic [12], so the situation might become clearer in the future. However, current users have to face the problem. For instance, the C99 standard states [13, Section 5.2.4.2.2]:

This work is partly supported by the TaMaDi project of the French *Agence Nationale de la Recherche*

É. Martin-Dorel
Inria Sophia Antipolis - Méditerranée, Marelle team,
2004 route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex, France
E-mail: erik.martin-dorel@ens-lyon.org

G. Melquiond
Inria Saclay–Île-de-France, Toccata team, LRI Lab., CNRS
Bât. 650, Univ. Paris Sud, 91405 Orsay Cedex, France E-mail: guillaume.melquiond@inria.fr

J.-M. Muller
CNRS, lab. LIP, Inria Aric team, Université de Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France
E-mail: jean-michel.muller@ens-lyon.fr

> *the values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type.*

To simplify, assume the various expressions of a program are of the same type. Two phenomenons may occur when a wider format is available in hardware (a typical example is the *double-extended format* available on *Intel x86*-compatible processors, for variables declared in the double-precision/binary64 format):

- For *temporary* values such as the result of "`a+b`" in the expression "`d = (a+b)*c`", it is not clear in which format they are computed.
- Explicitly *assigned* values may be first computed in the wider format, and then rounded to their destination format. This sometimes leads to a subtle problem called *double rounding* in the literature. This problem is illustrated by the following example.

Example 1.1 Consider the following C program [19]:

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

Notice that `a` and `b` are exactly representable in the binary64 format. Depending on the processor and the compilation options, we will either obtain

`3.6893488147419103232e+19`

or

`3.6893488147419111424e+19`,

which is the double-precision/binary64 number closest to the product `ab` (i.e., the value that should, ideally, be returned). Let us explain this. The exact value of `ab` is

$$36893488147419107329,$$

whose binary representation is

$$\overbrace{\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}0000000000}^{64\text{ bits}}01.$$

If the product is first rounded to the double-extended format, we get (in binary)

$$\overbrace{\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}0000000000}^{64\text{ bits}}\times 2^2.$$

Then, if that intermediate value is rounded to the double-precision destination format, this gives (using the round-to-nearest-even rounding mode)

$$\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}\times 2^{13}$$
$$= 36893488147419103232_{10}.$$

In general, using a greater precision is innocuous. However, it may make the behavior of some numerical programs difficult to predict. Interesting examples are given by Monniaux [18].

Most compilers offer options that avoid this problem. For instance, on a Linux Debian Etch 64-bit Intel platform, with GCC, the

```
-march=pentium4 -mfpmath=sse
```

compilation switches force the results of operations to be computed and stored in the 64-bit Streaming SIMD Extension (SSE) registers. However, such solutions have drawbacks:

- they significantly restrict the portability of numerical programs: e.g., it is difficult to make sure that one will always use algorithms such as 2Sum or Fast2Sum (see Section 2.7), in a large code, with the right compilation switches;
- they may have a bad impact on the performances of programs, as well as on their accuracy, since in most parts of a numerical program, it is in general more accurate to perform the intermediate calculations in a wider format.

Interestingly enough, as shown by Boldo and Melquiond, double rounding could be avoided if the wider precision calculations were implemented in a special rounding mode called *rounding to odd* [5]. Unfortunately, as we are writing this paper, rounding to odd is not implemented in floating-point units.

With the four arithmetic operations and the square root, one may easily find conditions on the precision of the wider format under which double roundings are innocuous. Such conditions have been explicited by Figueroa [8,9], who mentions that they probably have been given by Kahan in a 1998 lecture. For instance, in binary floating-point arithmetic, if the "target" format is of precision $p \geq 4$ and the wider format is of precision $p+p'$, double roundings are innocuous for addition if $p' \geq p+1$, for multiplication and division if $p' \geq p$, and for square root if $p' \geq p+2$. In the most frequent case ($p = 53$ and $p' = 11$), these conditions are not satisfied. Double roundings may also cause a problem in binary to decimal conversions. Solutions are given by Goldberg [10], and by Cornea et al. [6].

Hence, it is of interest to examine which properties of some basic computer arithmetic building blocks remain true when double roundings may occur. When these properties suffice, the obtained programs are more portable and "robust".

When the rounding mode (or direction) is toward $+\infty$, $-\infty$, or 0, one may easily check that double rounding cannot change the result of a calculation. As a consequence, in this paper, we will focus on "round to nearest" only.

This paper is organized as follows:

- Section 2 defines some notation, recalls the standard "epsilon model" for bounding errors of floating-point operations, and the classical 2Sum and Fast2Sum algorithms.
- Section 3 gives some preliminary remarks that will be useful later on.
- Section 4 analyzes the behavior of the Fast2Sum and 2Sum algorithms in the presence of double rounding. While they cannot always return the exact error of a floating-point addition, we show that they return the floating-point number *nearest* that error, under mild conditions.

– Section 5 analyzes the behavior of Veltkamp/Dekker's splitting algorithm in the presence of double rounding. That splitting algorithm allows one to express a precision-p floating-point number as the sum of a precision-s number and a precision-$(p-s)$ number. This is an important step of Dekker's multiplication algorithm. This is also useful for some summation algorithms.
– Fast2Sum and 2Sum are basic building blocks of many summation algorithms. In Section 6, we give some implications of the results obtained in the previous two sections to the behavior of these algorithms.

2 General assumptions, notation and background material

Throughout the paper, we assume that two binary floating-point formats are available on the same system:

– a precision-p "target" format with minimum exponent $e_{\min}$ and subnormal numbers,
– and a precision-$(p+p')$ wider binary "internal" format, whose exponent range is large enough to ensure that all floating-point numbers (normal or subnormal) of the target format are normal numbers of the wider format.

We also assume that no overflow occurs in the calculations.

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

$$x = M \cdot 2^{e-p+1}, \tag{2.1}$$

where M and e are integers such that

$$\begin{cases} |M| \leq 2^p - 1 \\ e_{\min} \leq e \leq e_{\max} \end{cases} \tag{2.2}$$

The number M of largest absolute value such that (2.1) and (2.2) hold is called the *integral significand* of x, and (when $x \neq 0$) the corresponding value of e is called the *exponent* of x. By convention, the exponent of zero is $e_{\min}$.

In the following, we denote by $\mathbb{F}_p$ the set of the precision-p binary FP numbers with minimum exponent $e_{\min}$ (i.e., the numbers of the "target" format).

2.1 Even, odd, normal and subnormal numbers, underflow

Let $x \in \mathbb{F}_p$, and let M_x be its integral significand. We will say that x is *even* (resp. *odd*) if M_x is even (resp. odd). We will say that x is *normal* if $|M_x| \geq 2^{p-1}$. A FP number that is not normal is called *subnormal*. A subnormal number has exponent $e_{\min}$, and its absolute value is less than $2^{e_{\min}}$. Zero is a subnormal number.

Concerning underflow, we will follow here the rule for raising the underflow flag of the default exception handling of the IEEE 754-2008 standard [12], and say that an operation induces an underflow when the result is both subnormal and inexact.

2.2 Roundings

$\mathrm{RN}_k(u)$ means *u rounded to the nearest precision-k FP number*, assuming round to nearest *even*: if u is exactly halfway between two consecutive precision-k FP numbers, $\mathrm{RN}_k(u)$ is the one of these two numbers that is even. When k is omitted, it means that $k = p$.

We say that $\circ$ is a *faithful rounding* if (i) when x is a FP number, $\circ(x) = x$, and (ii) when x is not a FP number, $\circ(x)$ is one of the two FP numbers that surround x.

We also say that a number *x fits in k bits* if $x \in \mathbb{F}_k$.

2.3 ulp notation, midpoints

If $|x| \in [2^e, 2^{e+1})$, with $e \leq e_{\max}$, we define $\mathrm{ulp}_p(x)$ as the number $2^{\max(e,e_{\min})-p+1}$. When there is no ambiguity on the considered precision, we omit the "p" and just write "$\mathrm{ulp}(x)$". Roughly speaking, $\mathrm{ulp}(x)$ is the distance between two consecutive FP numbers in the neighborhood of x; but such a definition lacks rigor when $|x|$ is near a power of 2.

A *precision-p midpoint* is a number exactly halfway between two consecutive precision-p FP numbers. If x and y are real numbers such that there is no midpoint between x and y then $\mathrm{RN}(x) = \mathrm{RN}(y)$. Notice that:

- if x is a nonzero FP number, and if $|x|$ is not a power of 2, then the two midpoints that surround x are $x - (1/2)\,\mathrm{ulp}(x)$ and $x + (1/2)\,\mathrm{ulp}(x)$;
- if $|x|$ is a power of 2 strictly larger than $2^{e_{\min}}$ and less than or equal to $2^{e_{\max}}$ then the two midpoints that surround x are $x - \mathrm{sign}(x) \cdot (1/4)\,\mathrm{ulp}(x)$ and $x + \mathrm{sign}(x) \cdot (1/2)\,\mathrm{ulp}(x)$.

In the following, we denote by $\mathbb{M}_p$ the set of the precision-p binary FP midpoints.

2.4 Double roundings and double rounding slips

When the arithmetic operation $x \top y$ appears in a program, we will say that:

- a *double rounding* occurs if what is actually performed is

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

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

In Example 1.1, the number obtained by rounding `a·b` to the widest format was a midpoint of the target format. This is the only case where double rounding slips may occur. More precisely,

Property 2.1 Let $\top$ be any arithmetic operation, and assume $p' \geq 1$. If a double rounding slip occurs when evaluating $a \top b$ then $\mathrm{RN}_{p+p'}(a \top b) \in \mathbb{M}_p$.

Proof If $\mathrm{RN}_p(a \top b)$ is not equal to $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(a \top b))$, then there is a precision-p midpoint, say μ, between $a \top b$ and $\mathrm{RN}_{p+p'}(a \top b)$. Since μ fits in $p+1$ bits, it is a precision-$(p+p')$ FP number. By definition $\mathrm{RN}_{p+p'}(a \top b)$ is a precision-$(p+p')$ FP number nearest $a \top b$, so $\mathrm{RN}_{p+p'}(a \top b)$ is equal to μ and thus it is a precision-p midpoint. □

2.5 The "standard model", or "ε-model"

In the FP literature, many properties are shown using the "standard model", or "ε-model", i.e., the fact that unless underflow occurs, the computed result of an arithmetic operation $a \top b$ satisfies:

$$\mathrm{RN}(a \top b) = (a \top b)(1+\varepsilon_1) = \frac{a \top b}{1+\varepsilon_2}, \tag{2.3}$$

where $|\varepsilon_1|, |\varepsilon_2| \leq u$, with $u = 2^{-p}$. When double rounding may occur, we get a similar property by using the fact that two consecutive roundings, one in precision $p+p'$ and one in precision p, are performed:

Property 2.2 (ε-model with double roundings) Let $a, b \in \mathbb{F}_p$, and let $\top \in \{+, -, \times, \div\}$.

- In the absence of underflow,

$$\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a \top b)\right) = (a \top b) \cdot (1+\varepsilon_1) = \frac{a \top b}{1+\varepsilon_2}, \tag{2.4}$$

  where $|\varepsilon_1|, |\varepsilon_2| \leq u'$, with $u' = 2^{-p} + 2^{-p-p'}$;
- For $\top = +$ or $\top = -$, (2.4) holds even in the presence of underflow.

Proof If no double rounding slip occurs, (2.3) holds, so that Property 2.2 holds as well. Now, assume that a double rounding slip occurs, and define a FP number σ as $\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a \top b)\right)$. We have

$$|(a \top b) - \mathrm{RN}_{p+p'}(a \top b)| \leq \frac{1}{2}\,\mathrm{ulp}_{p+p'}(a \top b) = 2^{-p'-1}\,\mathrm{ulp}_p(a \top b) \tag{2.5}$$

and

$$|\mathrm{RN}_{p+p'}(a \top b) - \sigma| \leq \frac{1}{2}\,\mathrm{ulp}_p\left(\mathrm{RN}_{p+p'}(a \top b)\right). \tag{2.6}$$

Since a double rounding slip occurred, Property 2.1 implies that $\mathrm{RN}_{p+p'}(a \top b) \in \mathbb{M}_p$. Therefore, there is no power of 2 between $a \top b$ and $\mathrm{RN}_{p+p'}(a \top b)$, so that $\mathrm{ulp}_p\left(\mathrm{RN}_{p+p'}(a \top b)\right) = \mathrm{ulp}_p(a \top b)$. This, combined with (2.5) and (2.6), gives

$$|(a \top b) - \sigma| \leq \left(\frac{1}{2} + 2^{-p'-1}\right)\mathrm{ulp}_p(a \top b). \tag{2.7}$$

- Since $\mathrm{ulp}(a \top b) \leq |a \top b| \cdot 2^{-p+1}$, we have $|(a \top b) - \sigma| \leq (2^{-p} + 2^{-p-p'}) \cdot |a \top b|$, from which we deduce

$$\sigma = (a \top b) \cdot (1+\varepsilon_1), \text{ with } |\varepsilon_1| \leq u'.$$

– Since $\mathrm{ulp}(a \top b) = \mathrm{ulp}_p(\mathrm{RN}_{p+p'}(a \top b)) \leq |\sigma 2^{-p+1}|$, we have $|(a \top b) - \sigma| \leq (2^{-p} + 2^{-p-p'}) \cdot |\sigma|$, from which we deduce

$$\sigma = (a \top b)/(1+\varepsilon_2), \text{ with } |\varepsilon_2| \leq u'.$$

The second part of Property 2.2 is straightforward. Indeed, when the sum or difference of two precision-p FP numbers is subnormal, that sum or difference is a precision-p FP number. So $\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a \pm b)\right) = (a \pm b)$.

□

When p' is large enough, the bound u' is only slightly larger than u, therefore most properties that can be shown using the ε-model only, remain true in the presence of double roundings (possibly with slightly larger error bounds).

2.6 u, u', γ_k, and γ' notations

In [11, page 67], Higham defines notations γ_k that are useful in error analysis. We slightly adapt them to the context of double rounding.

For any integer k, define

$$\gamma_k = \frac{ku}{1-ku},$$

and

$$\gamma'_k = \frac{ku'}{1-ku'},$$

where $u = 2^{-p}$ and $u' = 2^{-p} + 2^{-p-p'}$.

2.7 The 2Sum, Fast2Sum algorithms

The Fast2Sum algorithm was first introduced by Dekker [7]. It allows one to compute the error of a floating-point addition. Without double rounding, that algorithm is

Algorithm 1 (Fast2Sum(a,b))
$s \leftarrow \mathrm{RN}(a+b)$
$z \leftarrow \mathrm{RN}(s-a)$
$t \leftarrow \mathrm{RN}(b-z)$

We have the following result.

Theorem 2.1 (Fast2Sum algorithm ([7], and Theorem C of [16], page 236, adapted to the case of binary arithmetic)) *Let $a, b \in \mathbb{F}_p$, with exponents e_a, e_b respectively, be given. Assume $e_a \geq e_b$. Then the results of Algorithm 1 applied to a and b satisfy*

$$s = \mathrm{RN}(a+b) \quad \textit{and} \quad s+t = a+b.$$

Moreover, $z = s-a$ and $t = b-z$, that is, no rounding error occurs when computing z and t.

When we do not know in advance whether $e_a \geq e_b$ or not, it may be preferable to use the following algorithm, due to Knuth [16] and Møller [17].

Algorithm 2 (2Sum(a,b))
$s \leftarrow \mathrm{RN}(a+b)$
$a' \leftarrow \mathrm{RN}(s-b)$
$b' \leftarrow \mathrm{RN}(s-a')$
$\delta_a \leftarrow \mathrm{RN}(a-a')$
$\delta_b \leftarrow \mathrm{RN}(b-b')$
$t \leftarrow \mathrm{RN}(\delta_a+\delta_b)$

Knuth has shown that, if a and b are normal FP numbers and no underflow occurs, then $a+b=s+t$. Boldo et al. [4] have shown that the property still holds in presence of subnormal numbers and underflows. They have formally checked the proof of the following theorem with the Coq proof assistant [1] to ensure that no corner cases have been missed.

Theorem 2.2 *Let $a,b \in \mathbb{F}_p$. The results of Algorithm 2 applied to a and b satisfy*

$$s = \mathrm{RN}(a+b) \quad \textit{and} \quad s+t = a+b.$$

3 Some preliminary remarks

It is well known that the error of a rounded-to-nearest FP addition of two precision-p numbers is a precision-p number (it is precisely that error that is computed by Algorithms 1 and 2). When double rounding slips occur, the results of sums are slightly different from rounded to nearest sums. This difference, although it is very small, sometimes suffices to make the error not representable in FP arithmetic. More precisely, we have the following property.

Remark 3.1 If $p' \geq 1$ and $p' \leq p$, then there exist $a,b \in \mathbb{F}_p$ such that the number

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

satisfies

$$a+b-s \notin \mathbb{F}_p.$$

Proof To show this, it suffices to consider

$$a = 1\underbrace{xxxx\cdots x}_{p-3 \text{ bits}}01$$

where $xxxx\cdots x$ is any $(p-3)$-bit bit-chain. Number a is a p-bit integer, thus $a \in \mathbb{F}_p$. Also consider

$$b = 0.0\underbrace{111111\cdots 1}_{p \text{ ones}} = \frac{1}{2} - 2^{-p-1}.$$

Number b is equal to $(2^p-1)\cdot 2^{-p-1}$, thus $b \in \mathbb{F}_p$. We have

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

so that, if $1 \le p' \le p$,

$$c = \mathrm{RN}_{p+p'}(a+b) = 1xxxx\cdots x01.100\cdots 0.$$

The "round to nearest even" rule thus implies

$$s = \mathrm{RN}_p(c) = 1xxxx\cdots x10 = a+1.$$

Therefore,

$$s-(a+b) = a+1-(a+\frac{1}{2}-2^{-p-1}) = \frac{1}{2}+2^{-p-1} = 0.\underbrace{10000\cdots 01}_{p+1 \text{ bits}},$$

which does not belong to $\mathbb{F}_p$. □

Notice that the condition "$p' \le p$" in Remark 3.1 is necessary. More precisely, it is shown in [8,9] that if $p' \ge p+1$ then a double rounding slip cannot occur when computing $a+b$, i.e., we always have $\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a+b)\right) = \mathrm{RN}_p(a+b)$.

Remark 3.2 Assume that we compute $w = \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a+b))$, where $a,b \in \mathbb{F}_p$, with exponents e_a and e_b, respectively. Assume $e_a \ge e_b$ and $p' \ge 2$. If a double rounding slip occurs, then

$$e_a - p - 1 \le e_b \le e_a - p', \tag{3.1}$$

and

$$e_w \ge e_b + p' + 1. \tag{3.2}$$

Proof (First part: Equation (3.1)*)*
First, if $e_a - p - 1 > e_b$ (i.e., $e_a - p - 2 \ge e_b$), then

$$|b| < 2^{e_b+1} \le 2^{e_a-p-1} = \frac{1}{4}\,\mathrm{ulp}(a).$$

Also, $p' \ge 2$ implies that $a - \frac{1}{4}\,\mathrm{ulp}(a)$ and $a + \frac{1}{4}\,\mathrm{ulp}(a)$ are precision-$(p+p')$ FP numbers. Therefore,

$$a - \frac{1}{4}\,\mathrm{ulp}(a) \le \mathrm{RN}_{p+p'}(a+b) \le a + \frac{1}{4}\,\mathrm{ulp}(a).$$

As a consequence, if $|a|$ is not a power of 2, then $\mathrm{RN}_{p+p'}(a+b)$ cannot be a precision-p midpoint, which means that no double rounding slip occurs, according to Property 2.1.

So we are left with the case $|a| = 2^{e_a}$. Then $|a| - \frac{1}{4}\,\mathrm{ulp}(a)$ is a midpoint, and it is the only one between $|a| - \frac{1}{4}\,\mathrm{ulp}(a)$ and $|a| + \frac{1}{4}\,\mathrm{ulp}(a)$. Since $|b| < \frac{1}{4}\,\mathrm{ulp}(a)$, $a+b$ is between the midpoint $\mu = a - \mathrm{sign}(a)\cdot\frac{1}{4}\,\mathrm{ulp}(a)$ and $\mu' = a + \mathrm{sign}(a)\cdot\frac{1}{4}\,\mathrm{ulp}(a)$. Since $\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(\mu)\right) = a$ (due to the round-to-nearest *even* rounding rule) and

$\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(\mu')\right) = a$, the monotonicity of rounding functions ensures that we have $\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a+b)\right) = a = \mathrm{RN}(a+b)$. So no double rounding slip occurs in that case either.

Second, if $e_b > e_a - p'$, then $a+b$ can be written $2^{e_b-p+1}\left(2^{e_a-e_b}M_a + M_b\right)$, where M_a and M_b are the integral significands of a and b, and the integer $2^{e_a-e_b}M_a + M_b$ satisfies

$$\left|2^{e_a-e_b}M_a + M_b\right| \leq 2^{p'-1}(2^p-1) + (2^p-1) \leq 2^{p+p'} - 1.$$

Therefore $a+b \in \mathbb{F}_{p+p'}$, so that no double rounding slip can occur.

Proof (Second part: Equation (3.2)*)*
Let k be the integer such that $2^k \leq |a+b| < 2^{k+1}$. The monotonicity of the rounding functions implies that

$$2^k \leq \left|\mathrm{RN}_p(\mathrm{RN}_{p+p'}(a+b))\right| \leq 2^{k+1},$$

therefore, e_w is equal to k or $k+1$. Since $a+b$ does not fit into $p+p'$ bits (otherwise there would be no double rounding slip) and $a+b$ is a multiple of 2^{e_b-p+1}, we deduce that $\mathrm{ulp}_{p+p'}(a+b) > 2^{e_b-p+1}$, which implies that $k-p-p'+1 > e_b-p+1$. Therefore $e_w \geq k > e_b + p'$, which concludes the proof. □

An immediate consequence of Property 2.1 (due to the round-to-nearest-even rule) is the following.

Remark 3.3 Assume $p' \geq 1$. If a double rounding slip occurs when evaluating $a \top b$, then the returned result $\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a \top b)\right)$ is an even FP number.

In our proofs, we will also frequently use the following well-known result.

Remark 3.4 (Sterbenz' Lemma [29]) If a and b are positive elements of $\mathbb{F}_p$, and

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

then $a-b \in \mathbb{F}_p$, which implies that it will be computed exactly, whatever the rounding.

The following remark is directly adapted from Lemma 4 in Shewchuk's paper [28].

Remark 3.5 Let $a, b \in \mathbb{F}_p$. Let

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

If $|s| < \min\{|a|, |b|\}$ then $s = a+b$.

The proof can be found in [28, page 311].

Finally, the following result will be used in the proofs of Theorems 4.1 and 4.2.

Remark 3.6 Let $a, b \in \mathbb{F}_p$, and define $s = \mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a+b)\right)$. The number $a+b-s$ fits in at most $p+2$ bits, so that as soon as $p' \geq 2$, we have

$$\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a+b-s)\right) = \mathrm{RN}_p(a+b-s). \tag{3.3}$$

Proof Without loss of generality, we assume $e_a \geq e_b$. We already know that if no double rounding slip occurs when computing s, then $a+b-s \in \mathbb{F}_p$. In such a case, (3.3) is obviously true. So let us assume that $\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a+b)\right) \neq \mathrm{RN}_p(a+b)$. Equation (3.1) in Remark 3.2 implies therefore that $e_b \geq e_a - p - 1$.

Since a and b are both multiple of 2^{e_b-p+1}, we deduce that $a+b-s$ too is a multiple of 2^{e_b-p+1}. Also, since $|a|$ and $|b|$ are less than $(2^p-1)\cdot 2^{e_a-p+1}$, $|a+b|$ is less than $(2^p-1)\cdot 2^{e_a-p+2}$. Since rounding functions are monotonic and $(2^p-1)\cdot 2^{e_a-p+2} \in \mathbb{F}_p$, $|s|$ is less than or equal to $(2^p-1)\cdot 2^{e_a-p+2}$. Therefore $e_s \leq e_a+1$.

From all this, we deduce that $a+b-s$ is a multiple of 2^{e_b-p+1} of absolute value less than or equal to

$$2^{-p+e_s} + 2^{-p-p'+e_s} \leq 2^{-p+e_a+1} + 2^{-p-p'+e_a+1} \leq 2^{e_b+2} + 2^{e_b-p'+2}.$$

Thus $a+b-s$ fits in at most $p+2$ bits. Therefore, $\mathrm{RN}_{p+p'}(a+b-s) = a+b-s$ as soon as $p' \geq 2$. □

4 Behavior of Fast2Sum and 2Sum in the presence of double rounding

4.1 Fast2Sum and double rounding

Remark 3.1 implies that Algorithms Fast2Sum and 2Sum cannot always return the exact value of the error when $\mathrm{RN}(a+b)$ is replaced with $\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a+b)\right)$, i.e., when a double rounding occurs, because that error is not always a FP number. And yet, we may try to bound the difference between the error and the returned number t. Let us analyze how algorithm Fast2Sum behaves when double roundings are allowed. Consider the following algorithm,

Algorithm 3 (Fast2Sum-with-double-rounding(a,b)**)**
$s \leftarrow \mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(a+b)\right)$ *or* $\mathrm{RN}_p(a+b)$
$z \leftarrow \circ(s-a)$
$t \leftarrow \mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(b-z)\right)$ *or* $\mathrm{RN}_p(b-z)$

where $\circ$ is any faithful rounding (for instance, $\circ(x)$ can be either $\mathrm{RN}_p(x)$ or $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(x))$). We are going to see that $s-a \in \mathbb{F}_p$, so that $\circ(s-a) = s-a$.

Theorem 4.1 *Assume $p \geq 3$ and $p' \geq 2$. Let $a,b \in \mathbb{F}_p$ with exponents e_a and e_b respectively, be given. Assume $e_a \geq e_b$. Then the values z and t computed by Algorithm 3 satisfy*

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

Proof We already know from Theorem 2.1 that, if there is no double rounding slip at line 1 of Algorithm 3, lines 2 and 3 of the algorithm are executed without errors. Therefore, we only need to consider the case when a double rounding slip occurs at line 1. Assuming $p' \geq 2$, we deduce from Remark 3.2 that $e_b \leq e_a - p' \leq e_a - 2$. This implies $|b| \leq |a/2|$, from which we deduce that a and s have the same sign,

and that $|a/2| \leq |s| \leq |2a|$. According to Sterbenz' Lemma (Remark 3.4), $s-a \in \mathbb{F}_p$. Therefore, $z = s-a$, and $b-z = a+b-s$. If there is no double rounding at line 3 of Algorithm 3, we immediately deduce $t = \mathrm{RN}_p(a+b-s)$. Otherwise we have $t = \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a+b-s))$, from which we obtain $t = \mathrm{RN}_p(a+b-s)$ using Remark 3.6. □

4.2 2Sum and double roundings

We will now analyze the following algorithm.

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

(1) $s \leftarrow \square(a+b)$
(2) $a' \leftarrow \square(s-b)$
(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* $\mathrm{RN}_p(\delta_a+\delta_b)$

where $\square(x)$ means either $\mathrm{RN}_p(x)$ or $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(x))$ (but it is mandatory that the *same* rounding be applied at lines (1) and (2) of the algorithm), and $\circ(x)$ is either $\mathrm{RN}_p(x)$, $\mathrm{RN}_{p+p'}(x)$, or $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(x))$, or any faithful rounding.

Theorem 4.2 *Assume $p \geq 4$ and $p' \geq 2$. Let $a,b \in \mathbb{F}_p$. The results of Algorithm 4 applied to a and b satisfy:*

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

We will focus on proving that $t = \mathrm{RN}_p(a+b-s)$: when there is no double rounding slip when computing s, we already know that $a+b-s$ is a FP number, so that $\mathrm{RN}_p(a+b-s) = a+b-s$. Before giving a proof of Theorem 4.2, let us raise the following point:

Remark 4.1 Assuming $p' \geq 2$, if the variables a' and b' of Algorithm 4 satisfy $a' = a$ and $b' = s-a'$, then $t = \mathrm{RN}(a+b-s)$.

Proof If $a' = a$ then $\delta_a = 0$. Also, $b' = s-a' = s-a$ implies $b-b' = a+b-s$, so that $\delta_b = \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a+b-s))$. This gives

$$t = \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a+b-s)) = \mathrm{RN}_p(a+b-s),$$

using Remark 3.6. □

Let us now prove Theorem 4.2. We will assume that $\square$ is $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(\cdot))$, since when $\square$ is RN_p, the classical proof [7] (with no double roundings) applies.

Proof (First case: if $|b| \geq |a|$)
In that case, lines (1), (2), and (4) of Algorithm 4 constitute Algorithm 3, called with input values (b,a). Therefore (from Theorem 4.1), the computation of line (2) is exact, which implies $a' = s - b$ and $\delta_a = \mathrm{RN}_p(a+b-s)$. Also, $a' = s-b$ implies $s - a' = b$, so that $b' = b$ and $\delta_b = 0$. All this implies $t = \mathrm{RN}_p(a+b-s)$.

Proof (Second case: if $|b| < |a|$ and $|s| < |b|$)
In that case, Remark 3.5 applies: $s = a+b$, which implies $a' = a$, $b' = b$, and $\delta_a = \delta_b = t = 0$.

Proof (Third case: if $|b| < |a|$ and $|s| \geq |b|$)
In that case:

- From Property 2.2, we have $s = (a+b)\cdot(1+\varepsilon_1)$ and $a' = (s-b)\cdot(1+\varepsilon_2)$, with $|\varepsilon_1|, |\varepsilon_2| \leq u'$, from which we easily deduce

$$a' = (a + a\varepsilon_1 + b\varepsilon_1)\cdot(1+\varepsilon_2) = a\cdot(1+\varepsilon_3),$$

  with $|\varepsilon_3| \leq 3u' + 2u'^2$. A consequence is that as soon as $p \geq 4$ and $p' \geq 1$, $|a/2| \leq |a'| \leq |2a|$, and a and a' have the same sign. Hence, from Remark 3.4, $a - a' \in \mathbb{F}_p$, which implies $\delta_a = a - a'$. Moreover, we have $e_{a'} \leq e_a + 1$, which will be useful to establish (4.1).
- Lines (2), (3), and (5) constitute Algorithm 3, called with input values $(s,-b)$. This implies $b' = s - a'$ and $\delta_b = \mathrm{RN}_p(a' - (s-b))$. Moreover, Theorem 4.1 implies that $\delta_b = a' - (s-b)$ if no double rounding slip occurred at line (2). In such a case, $\delta_a + \delta_b = a + b - s$, which implies $t = \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a+b-s)) = \mathrm{RN}_p(a+b-s)$ (using Remark 3.6).

So, in the following, we assume that a double rounding slip occurred at line (2), i.e., when computing $s-b$. This implies from Remark 3.3 that a' is even. Notice that Equation (3.2) in Remark 3.2 implies that $e_b \leq e_{a'} - p' - 1$.

Also, we know that

- $\delta_a = a - a'$ and $b' = s - a'$;
- all variables $(s, a', b', \delta_a, \delta_b, t)$ are multiples of $2^{e_b - p + 1}$.

Since a double rounding slip occurred in line (2), we have

$$s - b = a' + i\cdot 2^{e_{a'}-p} + j\cdot\varepsilon,$$

where $i = \pm 1$ (or $\pm\frac{1}{2}$ in the case a' is a power of 2), $j = \pm 1$ and $0 \leq \varepsilon \leq 2^{e_{a'}-p-p'}$. Since $b' = s - a'$, we deduce

$$b' = b + i\cdot 2^{e_{a'}-p} + j\cdot\varepsilon,$$

hence

$$b - b' = -i\cdot 2^{e_{a'}-p} - j\cdot\varepsilon,$$

so that, since it is a multiple of 2^{e_b-p+1}, $b - b'$ fits in at most

$$(e_{a'} - p) - (e_b - p + 1) + 1 = e_{a'} - e_b \leq e_a - e_b + 1 \tag{4.1}$$

bits. As a result, if $e_a - e_b \leq p-1$ then $b - b' \in \mathbb{F}_p$, therefore $\delta_b = b - b'$. It follows that $\delta_a + \delta_b = a - a' + b - b' = a - a' + b - (s - a') = a + b - s$, so that $t = \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a+b-s)) = \mathrm{RN}_p(a+b-s)$ (using Remark 3.6).

Furthermore, if $e_a - e_b \geq p+2$, then one easily checks that $s = a' = a$, $b' = \delta_a = 0$, and $t = \delta_b = b$, which is the desired result.

Hence the last case that remains to be checked is the case $e_a - e_b \in \{p, p+1\}$. In that case,

- if a is not a power of 2, $s \in \{a^-, a, a^+\}$, where a^- and a^+ are the floating-point predecessor and successor of a;
- if a is a power of 2, s can also be equal to a^{--} (when $a > 0$) or a^{++} (when $a < 0$).

To simplify the presentation, we now assume $a > 0$. Otherwise, it suffices to change the signs of a and b. Notice that $s < a \Rightarrow a' \geq s$ (because in that case, $b < 0$), and $s > a \Rightarrow a' \leq s$.

1. *If a is not a power of* 2, then $s \in \{a^-, a, a^+\}$.
   - If $|b|$ is not of the form $\pm 2^{e_a - p} + \varepsilon$ with $|\varepsilon| \leq 2^{e_a - p - p'}$, then there are no double rounding slips in lines (1) and (2) of the algorithm.
   - Otherwise, if a is even, then (due to the round to nearest *even* rounding rule) $s = a$, and $a' = s = a$, therefore Remark 4.1 implies $t = \mathrm{RN}_p(a+b-s)$.
   - Otherwise, if a is odd, then $s = a^+$ or a^- and $a' = s$, so that $b' = 0$, which implies $\delta_b = b$ and $t = \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a - a' + b)) = \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a+b-s)) = \mathrm{RN}_p(a+b-s)$.
2. *If a is a power of* 2, i.e., $a = 2^{e_a}$. Notice that $e_b \in \{e_a - p, e_a - p + 1\}$ implies

$$\frac{1}{4}\,\mathrm{ulp}(a) \leq |b| < \mathrm{ulp}(a).$$

   - If $b \geq 0$, then s is equal to a, or a^+.
     - If $s = a^+ = a + \mathrm{ulp}(a)$ then

$$a + \mathrm{ulp}(a) - \mathrm{ulp}(a) < s - b \leq a + \mathrm{ulp}(a) - \frac{1}{4}\,\mathrm{ulp}(a),$$

       therefore (since $\square = \mathrm{RN}_p(\mathrm{RN}_{p+p'}(\cdot))$ is an increasing function),

$$a \leq \square(s-b) = a' \leq a + \mathrm{ulp}(a) = a^+.$$

       Since a' is even, we do not need to consider the case $a' = a^+$. If $a' = a$ then Remark 4.1 implies $t = \mathrm{RN}_p(a+b-s)$.
     - if $s = a$ then $\mathrm{RN}_{p+p'}(a+b) \leq a + \frac{1}{2}\,\mathrm{ulp}(a)$, hence

$$b \leq \frac{1}{2}\,\mathrm{ulp}(a) + 2^{e_a - p - p'}.$$

       In that case,

$$a - \frac{1}{2}\,\mathrm{ulp}(a) - 2^{e_a - p - p'} \leq s - b \leq a - \frac{1}{4}\,\mathrm{ulp}(a),$$

       hence,

$$a^- \leq \square(s-b) = a' \leq a.$$

       Since a' is even, we do not need to consider the case $a' = a^-$. If $a' = a$ then Remark 4.1 implies $t = \mathrm{RN}_p(a+b-s)$.

- If $b < 0$, then $a - \mathrm{ulp}(a) < a + b \le a - \frac{1}{4}\mathrm{ulp}(a)$, which implies $a^{--} = a - \mathrm{ulp}(a) \le s \le a$.
  - If $s = a$ then $\mathrm{RN}_{p+p'}(a+b) = a - \frac{1}{4}\mathrm{ulp}(a)$, hence $b \ge -\frac{1}{4}\mathrm{ulp}(a) - 2^{e_a-p-p'-1}$, therefore
  $$a < s - b \le a + \frac{1}{4}\mathrm{ulp}(a) + 2^{e_a-p-p'-1}.$$
  This implies
  $$a \le \mathrm{RN}_{p+p'}(s-b) \le a + \frac{1}{4}\mathrm{ulp}(a),$$
  so that $a' = \square(s-b) = a$. From Remark 4.1, we deduce that $t = \mathrm{RN}_p(a+b-s)$.
  - If $s = a^- = a - \frac{1}{2}\mathrm{ulp}(a)$, then
  $$a - \frac{3}{4}\mathrm{ulp}(a) < \mathrm{RN}_{p+p'}(a+b) < a - \frac{1}{4}\mathrm{ulp}(a),$$
  hence,
  $$-\frac{3}{4}\mathrm{ulp}(a) < b < -\frac{1}{4}\mathrm{ulp}(a).$$
  This implies
  $$a - \frac{1}{4}\mathrm{ulp}(a) < s - b < a + \frac{1}{4}\mathrm{ulp}(a),$$
  so that $a' = \square(s-b) = a$. From Remark 4.1, we deduce that $t = \mathrm{RN}_p(a+b-s)$.
  - If $s = a^{--} = a - \mathrm{ulp}(a)$ then $\mathrm{RN}_{p+p'}(a+b) \le a - \frac{3}{4}\mathrm{ulp}(a)$, therefore
  $$b \le -\frac{3}{4}\mathrm{ulp}(a) + 2^{e_a-p-p'-1},$$
  so that (since we also have $-\mathrm{ulp}(a) < b$),
  $$a - \frac{1}{4}\mathrm{ulp}(a) - 2^{e_a-p-p'-1} \le s - b < a.$$
  This implies
  $$a^- \le a' = \square(s-b) \le a.$$
  Since a' is even, we do not need to consider the case $a' = a^-$. If $a' = a$ then Remark 4.1 implies $t = \mathrm{RN}_p(a+b-s)$.

If a different rounding is applied at lines (1) and (2) of Algorithm 4, the returned result t may differ from $\mathrm{RN}(a+b-s)$. A counterexample is obtained by applying rounding RN_p at line (1), rounding $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(\cdot))$ at line (2), and choosing $a = 2^{p-1}+1$ and $b = 1/2 - 2^{-p-1}$.
An immediate consequence of Theorems 4.1 and 4.2 is as follows.

Corollary 4.1 *The values s and t returned by Algorithms 3 or 4 satisfy*

$$(s+t) = (a+b)(1+\eta),$$

with $|\eta| \le uu' = 2^{-2p} + 2^{-2p-p'}$.

Even when a double rounding slip occurred when computing s in Fast2Sum or 2Sum, $(a+b)-s$ will often be exactly representable. More precisely, we have the following result.

Remark 4.2 Assume $p' \geq 2$. Let $a, b \in \mathbb{F}_p$ with exponents e_a, e_b, respectively, be given. Assume $e_a \geq e_b$, and define

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

If $a+b-s \notin \mathbb{F}_p$, then $e_b = e_a - p - 1$.

Proof Remark 3.2 and the assumption $p' \geq 2$ imply

$$e_a - p - 1 \leq e_b \leq e_a - p' \leq e_a - 2,$$

therefore $|b| < |a/2|$, which implies $|b| < |s| < 2|a|$. Let $r = (a+b)-s$. We have

$$|r| < \mathrm{ulp}(s) \leq 2\,\mathrm{ulp}(a). \tag{4.2}$$

Also, since a, b, s, and therefore r are all multiple of $\mathrm{ulp}(b)$, there exists an integer M_r such that

$$r = M_r \cdot \mathrm{ulp}(b). \tag{4.3}$$

- If $e_b \geq e_a - p + 1$ then $\mathrm{ulp}(a)/\mathrm{ulp}(b) \leq 2^{p-1}$. Combined with (4.2) and (4.3), this gives $|M_r| \leq 2^p$, which implies $r \in \mathbb{F}_p$;
- if $e_b = e_a - p$ then $\mathrm{ulp}(a)/2 \leq |b| < \mathrm{ulp}(a)$. Therefore $a - \mathrm{ulp}(a) < a + b < a + \mathrm{ulp}(a)$, which implies $a - \mathrm{ulp}(a) \leq s \leq a + \mathrm{ulp}(a)$. We deduce (by separately handling the cases $b \leq 0$ and $b > 0$) that $|r| < \mathrm{ulp}(a)$. Combined with $\mathrm{ulp}(a) = 2^p\,\mathrm{ulp}(b)$, this implies $|M_r| < 2^p$, therefore $r \in \mathbb{F}_p$.

Therefore, if $a+b-s \notin \mathbb{F}_p$, then $e_b = e_a - p - 1$. □

The following consequence of Remark 4.2 will be of interest when discussing the Splitting algorithm of Rump, Ogita, and Oishi (useful for designing a summation algorithm):

Remark 4.3 If a is a power of 2 and $|b| \leq a$, then the values s and t returned by Algorithm 3 or Algorithm 4 satisfy $t = a+b-s$.

Proof We know that if $s = \mathrm{RN}_p(a+b)$, then $a+b-s \in \mathbb{F}_p$, so $t = \mathrm{RN}_p(a+b-s) = a+b-s$. Therefore, let us assume that a double rounding slip occurred while computing s:

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

and that $a+b-s \notin \mathbb{F}_p$. Remark 4.2 implies $e_b = e_a - p - 1$, so that

$$|b| < \frac{1}{2}\,\mathrm{ulp}(a).$$

– If $b > 0$ then the only case for which we may have a double rounding slip (according to Property 2.1, and the fact that $a+b < a+\frac{1}{2}\,\mathrm{ulp}_p(a) \Rightarrow \mathrm{RN}_{p+p'}(a+b) \leq a+\frac{1}{2}\,\mathrm{ulp}(a)$) is

$$\mathrm{RN}_{p+p'}(a+b) = 2^{e_a} + \frac{1}{2}\,\mathrm{ulp}(a).$$

– If $b < 0$ then then the only case for which we may have a double rounding slip (still according to Property 2.1, and the fact that $a+b > a-\frac{1}{2}\,\mathrm{ulp}(a) = a^-$, so that $\mathrm{RN}_{p+p'}(a+b) \geq a^-$) is

$$\mathrm{RN}_{p+p'}(a+b) = 2^{e_a} - \frac{1}{4}\,\mathrm{ulp}(a).$$

In both cases, the round-to-nearest-*even* rounding rule implies $s = 2^{e_a} = a$, so that $a+b-s = b$, which contradicts the assumption that $a+b-s \notin \mathbb{F}_p$. □

5 Splitting algorithms in the presence of double roundings

5.1 Veltkamp's splitting algorithm in the presence of double roundings

In some applications (e.g., to implement Dekker's multiplication algorithm [7]), we need to "split" a precision-p floating-point number x in two FP numbers x_h and x_ℓ such that, for a given $s \leq p-1$, x_h fits in $p-s$ bits, x_ℓ fits in s bits, and $x = x_h + x_\ell$.

Veltkamp's algorithm (Algorithm 5) can be used for that purpose [7]. It uses a constant C equal to 2^s+1 (which belongs to $\mathbb{F}_p$ as soon as $s \leq p-1$).

Algorithm 5 (Split(x,s): Veltkamp's algorithm)
Require: $C = 2^s+1$
$\gamma \leftarrow \mathrm{RN}(C \cdot x)$
$\delta \leftarrow \mathrm{RN}(x-\gamma)$
$x_h \leftarrow \mathrm{RN}(\gamma+\delta)$
$x_\ell \leftarrow \mathrm{RN}(x-x_h)$

Boldo [2] shows that for any precision p, the variables x_h and x_ℓ returned by the algorithm satisfy the desired properties. Moreover x_ℓ actually fits in $s-1$ bits, which is an important feature when one wishes to implement Dekker's product algorithm with a binary FP format of odd precision. Let us see what happens if double roundings may occur. More precisely, we will analyze the following algorithm:

Algorithm 6 (Split(x,s): Veltkamp's algorithm with possible double roundings)
Require: $C = 2^s+1$
(1) $\gamma \leftarrow \mathrm{RN}_p(C \cdot x)$ *or* $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(C \cdot x))$
(2) $\delta \leftarrow \mathrm{RN}_p(x-\gamma)$ *or* $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(x-\gamma))$
(3) $x_h \leftarrow \circ(\gamma+\delta)$
(4) $x_\ell \leftarrow \circ(x-x_h)$

where $\circ(x)$ can be either $\mathrm{RN}_p(x)$, $\mathrm{RN}_{p+p'}(x)$, or $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(x))$, or any faithful rounding.

We can no longer be sure that x_ℓ fits in $s-1$ bits (unless no double rounding slip occurs at step (2)). This is illustrated by the following example: assume $p = 11$, $p' = 3$, $s = 5$, and $x = 1041_{10} = 10000010001_2$. If we run Algorithm 6 with rounding $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(\cdot))$ at steps (1) and (2), we obtain $x_h = 1024_{10} = 10000000000_2$, and $x_\ell = 17_{10} = 10001_2$, which fits in 5 bits, but does not fit in 4 bits. However, we have the following result.

Theorem 5.1 *Assume $2 \le s \le p-2$, $p \ge 5$, and $p' \ge 2$. For any input value $x \in \mathbb{F}_p$, the values x_h and x_ℓ returned by Algorithm 6 satisfy:*

- $x = x_h + x_\ell$;
- *x_h fits in $p-s$ bits;*
- *x_ℓ fits in s bits.*

Proof Without loss of generality, we assume $x > 0$. We also assume that x is not a power of 2 (otherwise, the proof is straightforward). This gives

$$2^{e_x} + 2^{e_x-p+1} \le x \le 2^{e_x+1} - 2^{e_x-p+1},$$

where e_x is the FP exponent of x. Furthermore, one may easily check that when $x = 2^{e_x+1} - 2^{e_x-p+1}$, the algorithm returns a correct result:

- if $s+1 \le p'$, then no double rounding slip can occur when computing γ, we get $\gamma = 2^{e_x+s+1} + 2^{e_x+1} - 2^{e_x+s-p+2}$, $\delta = -(2^{e_x+s+1} - 2^{e_x+s-p+2})$, $x_h = 2^{e_x+1}$, and $x_\ell = -2^{e_x-p+1}$;
- if $s \ge p'$ and step (1) consists of $\gamma \leftarrow \mathrm{RN}_p(C \cdot x)$, then we also get $\gamma = 2^{e_x+s+1} + 2^{e_x+1} - 2^{e_x+s-p+2}$, $\delta = -(2^{e_x+s+1} - 2^{e_x+s-p+2})$, $x_h = 2^{e_x+1}$, and $x_\ell = -2^{e_x-p+1}$;
- and if $s \ge p'$ and step (1) involves a double rounding, then a double rounding slip occurs when computing γ, we get $\gamma = 2^{e_x+s+1} + 2^{e_x+1}$, $\delta = -2^{e_x+s+1}$, $x_h = 2^{e_x+1}$, and $x_\ell = -2^{e_x-p+1}$.

Therefore, in the following, we assume $2^{e_x} + 2^{e_x-p+1} \le x \le 2^{e_x+1} - 2^{e_x-p+2}$. Without difficulty, we find

$$\gamma = (2^s + 1) \cdot x + \varepsilon_1,$$

with $|\varepsilon_1| \le 2^{e_x+s-p+1} + 2^{e_x+s-p-p'+1}$. We have

$$\begin{aligned} |x - \gamma| &\le 2^s \cdot x + |\varepsilon_1| \\ &\le 2^s \left(2^{e_x+1} - 2^{e_x-p+2}\right) + 2^{e_x+s-p+1} + 2^{e_x+s-p-p'+1} \\ &< 2^{e_x+s+1} \end{aligned}$$

(as soon as $p' \ge 1$), from which we deduce

$$\delta = x - \gamma + \varepsilon_2,$$

with $|\varepsilon_2| \le 2^{e_x+s-p} + 2^{e_x+s-p-p'}$. Furthermore,

$$|x - \gamma| \ge 2^s(2^{e_x} + 2^{e_x-p+1}) - (2^{e_x+s-p+1} + 2^{e_x+s-p-p'+1}).$$

This implies

$$|x-\gamma| \geq 2^{e_x+s}(1-2^{-p-p'+1}).$$

Because of the monotonicity of rounding, as soon as $p' \geq 2$, we have $|\delta| \geq \mathrm{RN}_p[2^{e_x+s} \cdot (1-2^{-p-p'+1})] = 2^{e_x+s}$ (resp. $|\delta| \geq \mathrm{RN}_p(\mathrm{RN}_{p+p'}[2^{e_x+s} \cdot (1-2^{-p-p'+1})]) = 2^{e_x+s}$, depending on the rounding involved in step (2)). This implies that δ is a multiple of $2^{e_x+s-p+1}$.

Let us now focus on the computation of x_h. So far, we have obtained

$$-\delta = -x+\gamma-\varepsilon_2 = 2^s x+\varepsilon_1-\varepsilon_2$$

and

$$\gamma = (2^s+1)x+\varepsilon_1,$$

with

$$|\varepsilon_1| \leq 2^{e_x+s-p+1}+2^{e_x+s-p-p'+1} \leq (2^{s-p+1}+2^{s-p-p'+1}) \cdot x$$

and

$$|\varepsilon_2| \leq (2^{s-p}+2^{s-p-p'}) \cdot x.$$

Therefore

$$\frac{2^s}{2^s+1} \cdot \frac{1-2^{-p+2}-2^{-p-p'+2}}{1+2^{-p+1}+2^{-p-p'+1}} \leq \left|\frac{\delta}{\gamma}\right| \leq \frac{2^s}{2^s+1} \cdot \frac{1+2^{-p+2}+2^{-p-p'+2}}{1-2^{-p+1}-2^{-p-p'+1}},$$

so that as soon as $p \geq 5$, $p' \geq 1$, and $s \geq 2$, $|\gamma|$ and $|\delta|$ are within a factor 2 from each other. Hence, since γ and δ clearly have opposite signs, we deduce from Remark 3.4 that $x_h = \gamma+\delta$ is computed exactly. Also, since γ and δ are multiples of $2^{e_x+s-p+1}$, x_h is multiple of $2^{e_x+s-p+1}$ too. Moreover,

$$\begin{aligned} x_h = x+\varepsilon_2 &\leq \left(2^{e_x+1}-2^{e_x-p+1}\right)+\left(2^{e_x+s-p}+2^{e_x+s-p-p'}\right) \\ &< 2^{e_x+1}+2^{e_x+s-p+1}, \end{aligned}$$

which implies:

- either $x_h < 2^{e_x+1}$, in which case, since x_h is a multiple of $2^{e_x+s-p+1}$, it fits in $p-s$ bits;
- or $x_h \geq 2^{e_x+1}$, but the only possible value that is both multiple of $2^{e_x+s-p+1}$ and less than $2^{e_x+1}+2^{e_x+s-p+1}$ is 2^{e_x+1}, which fits in 1 bit.

Therefore, in any case, x_h fits in at most $p-s$ bits.

There now remains to focus on the computation of x_ℓ. Since $|x-x_h| = |\varepsilon_2| \leq 2^{e_x+s-p}+2^{e_x+s-p-p'} \leq x \cdot (2^{s-p}+2^{s-p-p'})$, we easily find that as soon as $s \leq p-2$, one can apply Remark 3.4 and deduce that $x_\ell = x-x_h$ is computed exactly. Hence, we have $x = x_h+x_\ell$. Furthermore,

$$|x_\ell| = |\varepsilon_2| \leq 2^{e_x+s-p}+2^{e_x+s-p-p'},$$

and (since both x and x_h are multiples of 2^{e_x-p+1}), x_ℓ is a multiple of 2^{e_x-p+1} too. From this we deduce that x_ℓ fits in s bits. □

If $s \leq p'+1$, there is no multiple of 2^{e_x-p+1} between 2^{e_x+s-p} and $2^{e_x+s-p}+2^{e_x+s-p-p'}$. In such a case, $x_\ell \leq 2^{e_x+s-p}$, and we deduce that x_ℓ fits in $s-1$ bits.

5.2 Application to the computation of exact products

A consequence of Theorem 5.1 is that if the precision p is an even number, then Dekker's product algorithm [7] can be used: the proof of Dekker's product given for instance in [19, pages 137–139] shows that all operations but those of the Veltkamp's splitting in Dekker's algorithm are exact operations, so that they are not hindered by double roundings.

Dekker's product algorithm requires 17 operations. If a fused multiply-add (FMA) instruction is available,[1] there is a much better solution for expressing the product of two FP numbers as the unevaluated sum of two FP numbers, which works even in the presence of double roundings. That solution can be traced back at least to Kahan [15]. One may find a good presentation in [21]. More precisely (once adapted to the context of double roundings), we get the following theorem.

Theorem 5.2 *Let $\circ$ denote either $\mathrm{RN}_p(\cdot)$, $\mathrm{RN}_{p+p'}(\cdot)$, or $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(\cdot))$, or any faithful rounding. Let $x, y \in \mathbb{F}_p$, with exponents e_x and e_y, respectively. If*

$$e_x + e_y \geq e_{\min} + p - 1$$

then the sequence of calculations

Algorithm 2MultFMA(x, y)
$\pi_h \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(xy))$ *or* $\mathrm{RN}_p(xy)$
$\pi_\ell \leftarrow \circ(xy - \pi_h)$

returns $\pi_h \in \mathbb{F}_p$ and $\pi_\ell \in \mathbb{F}_p$ such that $xy = \pi_h + \pi_\ell$.

The proof just uses the fact (see Theorem 2 in [3]) that for any faithful rounding $\odot$, $xy - \odot(xy) \in \mathbb{F}_p$, provided that $e_x + e_y \geq e_{\min} + p - 1$.

5.3 Rump, Ogita, and Oishi's Extractscalar splitting algorithm

In [27], Rump, Ogita, and Oishi introduce a splitting algorithm and use it for designing very accurate summation methods. We will now see that most properties of their *Extractscalar splitting algorithm* are preserved in the presence of double rounding. Rewritten with double roundings, their algorithm is as follows.

Algorithm 7 (Extractscalar-with-double-rounding(σ, x)**)**
Require: $\sigma = 2^k$
$s \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(x + \sigma))$
$y \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(s - \sigma))$
$x' \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(x - y))$

When there are no double roundings, Rump, Ogita, and Oishi show that, if the exponent of x satisfies $e_x \leq k$, then $y + x' = x$, y is a multiple of 2^{k-p}, and $|x'| \leq 2^{k-p}$. The reason behind this is that Algorithm 7 is a variant of Fast2Sum. When double roundings are allowed, we get the following result.

[1] The FMA instruction evaluates expressions of the form $xy + z$ with one final rounding only.

Property 5.1 (Rump, Ogita, and Oishi's Extractscalar splitting algorithm in the presence of double rounding) If $e_x \leq k$ then the values y and x' computed by Algorithm 7 satisfy $y + x' = x$, y is a multiple of 2^{k-p}, and $|x'| \leq 2^{k-p} + 2^{k-p-p'}$.

Proof Algorithm 7 is Fast2Sum-with-double-rounding(σ, x). Theorem 4.1 and Remark 4.3 (since σ is a power of 2) imply that $s + x' = \sigma + x$ and $y = s - \sigma$, so that $y + x' = x$.

Furthermore, s is a multiple of 2^{k-p} (this is obtained by considering that either $|x| \geq 2^{k-1}$, in which case x and therefore $x + \sigma$ are multiple of 2^{k-p}, or $|x| < 2^{k-1}$, in which case $x + \sigma > 2^{k-1}$, hence, $s \geq 2^{k-1}$, which implies that s is a multiple of 2^{k-p}).

An immediate consequence is that y is a multiple of 2^{k-p}. Now, the (possibly double rounded) computed value of s satisfies

$$-2^{k-p} - 2^{k-p-p'} \leq s - (x + \sigma) \leq 2^{k-p} + 2^{k-p-p'},$$

which implies

$$-2^{k-p} - 2^{k-p-p'} \leq x' \leq 2^{k-p} + 2^{k-p-p'}.$$

□

6 Consequences of Theorems 4.1 and 4.2 on summation algorithms

Many numerical problems require the computation of sums of a large number of FP numbers. Several *compensated summation* algorithms use, either implicitly or explicitly, the Fast2Sum or 2Sum algorithms [14,24,25,22,26]. Therefore, when double rounding may occur, it is of importance to know whether returning the FP number nearest the error of a FP addition (instead of that error itself) may have an influence on the behavior of these algorithms. There is a large literature on summation algorithms: the purpose of this section is not to examine all published algorithms, just to give a few examples.

6.1 The recursive sum algorithm and Kahan's compensated summation algorithm in the presence of double rounding

Before analyzing other summation methods, let us see what happens with the naive, "recursive sum" algorithm, rewritten with double rounding.

Algorithm 8
$r \leftarrow a_1$
for $i = 2$ *to* n ***do***
  $r \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(r + a_i))$
end for
return r

A straightforward adaptation of the proof for the error bound of the usual recursive sum algorithm without double rounding gives the following property.

Property 6.1 The final value of the variable r returned by Algorithm 8 satisfies

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

Without double roundings, the bound is $\gamma_{n-1} \sum_{i=1}^{n} |a_i|$. See Section 2.6 for a definition of notations γ_k and γ'_k.

Kahan's compensated summation algorithm, rewritten with double rounding, is as follows.

Algorithm 9 (Kahan compensated summation algorithm)
$s \leftarrow a_1$
$c \leftarrow 0$
for $i = 2$ *to* n ***do***
  $y \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(a_i - c))$
  $t \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(s + y))$
  $c \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(\mathrm{RN}_p(\mathrm{RN}_{p+p'}(t - s)) - y))$
  $s \leftarrow t$
end for
return s

Goldberg's proof for Kahan's algorithm [10] only uses the ε-model, so that adaptation to double rounding is straightforward (it suffices to replace u with u'), and we immediately deduce that the value s returned by Algorithm 9 satisfies

$$\left| s - \sum_{i=1}^{n} a_i \right| \leq \left(2u' + O(nu'^2)\right) \cdot \sum_{i=1}^{n} |a_i|$$

This makes Kahan's compensated summation algorithm very "robust": double roundings have little influence on the error bound. However, when $\sum_{i=1}^{n} |a_i|$ is very large compared to $|\sum_{i=1}^{n} a_i|$, the relative error of Kahan's compensated summation algorithm becomes large. A solution is to use Priest's *doubly compensated* summation algorithm [25]. For that algorithm, the excellent error bound $2u\,|\sum_{i=1}^{n} a_i|$ remains true even in the presence of double rounding (the proof essentially assumes faithfully rounded operations). However, it requires a preliminary sorting of the a_i's by magnitude.

In the following, we investigate the potential influence of double rounding on some sophisticated summation algorithms. For most of these algorithms, the proven error bounds (without double rounding) are of the form

$$\left| \text{computed sum} - \sum_{i=1}^{n} a_i \right| \leq u \cdot \left| \sum_{i=1}^{n} a_i \right| + \alpha \cdot \sum_{i=1}^{n} |a_i|.$$

Rump, Ogita, and Oishi exhibit a family of algorithms for which, without double rounding, α has the form $O(n^K 2^{-Kp})$. As we will see, that property will be (roughly) preserved when $K = 2$. However, for the more subtle algorithms (for which $K \geq 3$), double rounding may ruin that property.

6.2 Rump, Ogita and Oishi's cascaded summation algorithm in the presence of double rounding

The following algorithm was independently introduced by Pichat [23] and by Neumaier [20].

Algorithm 10 (Pichat-Neumaier summation algorithm)
$s \leftarrow a_1$
$e \leftarrow 0$
for $i = 2$ *to* n ***do***
  if $|s| \geq |a_i|$ ***then***
    $(s, e_i) \leftarrow \mathit{Fast2Sum}(s, a_i)$
  else
    $(s, e_i) \leftarrow \mathit{Fast2Sum}(a_i, s)$
  end if
  $e \leftarrow \mathrm{RN}(e + e_i)$
end for
return $\mathrm{RN}(s + e)$

To avoid tests, the algorithm of Pichat and Neumaier can be rewritten using the 2Sum algorithm. This gives the *cascaded summation* algorithm of Rump, Ogita, and Oishi [22]:

Algorithm 11 (Rump, Ogita, and Oishi's Cascaded Summation algorithm)
$s \leftarrow a_1$
$e \leftarrow 0$
for $i = 2$ *to* n ***do***
  $(s, e_i) \leftarrow \mathit{2Sum}(s, a_i)$
  $e \leftarrow \mathrm{RN}(e + e_i)$
end for
return $\mathrm{RN}(s + e)$

Both algorithms return the same result. In the following, we therefore focus on Algorithm 11 only. More precisely, we are interested here in analyzing the behavior of that algorithm, with double rounding allowed:

Algorithm 12 (Algorithm 11 with double rounding)
$s \leftarrow a_1$
$e \leftarrow 0$
for $i = 2$ *to* n ***do***
  $(s, e_i) \leftarrow \mathit{2Sum\text{-}with\text{-}double\text{-}rounding}(s, a_i)$
  $e \leftarrow \mathrm{RN}_p(\mathrm{RN}_{p+p'}(e + e_i))$
end for
return $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(s + e))$

We have the following result.

Theorem 6.1 *Assume $p \geq 8$, $p' \geq 4$, and $n-1 < \dfrac{1}{2u'}$. The final value σ returned by Algorithm 12 satisfies*

$$\left|\sigma - \sum_{i=1}^{n} a_i\right| \leq \left(2^{-p} + 2^{-p-p'} + 2^{-2p-p'}\right) \cdot \sum_{i=1}^{n} a_i + 2^{-2p} \cdot \left(4n^2 - 10n - 5\right) \cdot \left(1 + 2^{-p'+1} + \frac{3}{200}\right) \cdot \sum_{i=1}^{n} |a_i|.$$

The proof of Theorem 6.1 is very similar to the proof of the error bound of Algorithm 11(i.e., the bound of the classical, double-rounding-free, algorithm). In that classical case, the term in front of $\sum_{i=1}^{n} a_i$ is $u = 2^{-p}$, and the term in front of $\sum_{i=1}^{n} |a_i|$ is γ_{n-1}^2. Hence the Cascaded Summation algorithm is "robust" and can be used safely, even when double rounding may happen: the error bound is slightly larger but remains of the same order of magnitude.

However, more subtle algorithms, that return a more accurate result (assuming no double rounding) when $(\sum_{i=1}^{n} |a_i|)/|\sum_{i=1}^{n} a_i|$is very large, may be of less interest when double rounding may happen, unless we have some additional information on the input data that allows one to make sure there will be no problem. Consider for instance the K-fold summation algorithm of Rump, Ogita, and Oishi, defined as follows.

Algorithm 13 (VecSum(a), where $a = (a_1, a_2, \ldots, a_n)$)
$p \leftarrow a$
for $i = 2$ *to* n ***do***
  $(p_i, p_{i-1}) \leftarrow 2Sum(p_i, p_{i-1})$
end for
return p

Algorithm 14 (Rump, Ogita and Oishi's K-fold summation algorithm)
for $k = 1$ *to* $K-1$ ***do***
  $a \leftarrow VecSum(a)$
end for
$c = a_1$
for $i = 2$ *to* $n-1$ ***do***
  $c \leftarrow \mathrm{RN}(c + a_i)$
end for
return $\mathrm{RN}(a_n + c)$

If double roundings are not allowed, Rump, Ogita, and Oishi show that if $4nu < 1$, the final result σ returned by Algorithm 14 satisfies

$$\left|\sigma - \sum_{i=1}^{n} a_i\right| \leq (u + \gamma_{n-1}^2) \left|\sum_{i=1}^{n} a_i\right| + \gamma_{2n-2}^K \sum_{i=1}^{n} |a_i|. \tag{6.1}$$

If a double rounding slip occurs in the first call to VecSum, an error as large as $2^{-2p} \max |a_i|$ may be produced. Hence, it will not be possible to show a final error bound better than $2^{-2p} \max |a_i| \geq (2^{-2p}/n) \cdot \sum_{i=1}^{n} |a_i|$ when double roundings are allowed. In practice (since double rounding slips are not so frequent, and do not always

change the result of 2Sum when they occur), the K-fold summation algorithm will almost always return a result that satisfies a bound close to the one given by (6.1), but exceptions may occur. Consider the following example (with $n = 5$, but easily generalizable to any larger value of n):

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

and assume that Algorithm 14 is run with double rounding, with $1 \leq p' \leq p$. In the first addition of the first 2Sum of the first call to VecSum (i.e., when adding a_1 and a_2), a double rounding slip occurs, so that immediately after this 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 $(-1/2, 0, 0, 0, 1/2)$, so that Algorithm 14 will return 0 whatever the value of K, whereas the exact sum of the a_i's is -2^{-p-1}. Hence (since $\sum |a_i| = 2^p + 4 - 2^{-p-1} \approx 2^p$), the final error of Algorithm 14 is approximately $2^{-2p-1} \sum |a_i|$, whatever the value of K.

This example shows that, if we wish to get error bounds with a magnitude of the same order as the one given by (6.1) when using the K-fold summation algorithm with $K \geq 3$, we need to select compilation switches that prevent double rounding from occurring, unless we have additional information on the input data that allows one to show that Fast2Sum and 2Sum will return an exact result, even in the presence of double rounding.

Rump, Ogita, and Oishi suggest another summation algorithm that returns faithfully rounded sums when run without double rounding [27]. It is based on the splitting algorithm discussed in Section 5.3. Using Property 5.1, one may adapt their summation algorithm, so that it can return faithfully rounded sums, even in the presence of double rounding.

7 Conclusion and future work

We have considered the possible influence of double rounding on several algorithms of the floating-point literature: Fast2Sum, 2Sum, Veltkamp's splitting, 2MultFMA, and some summation algorithms. Although most of these algorithms do not behave exactly as when there are no double roundings, they still have interesting properties that can be exploited in an useful way. Depending on the applications, these properties may suffice, or specific compilation options should be chosen to prevent double rounding.

Proofs in computer arithmetic are somewhat complex: they are frequently based on the enumeration of many possible cases, so that one may very easily overlook one of these cases. To avoid this problem and get more confidence in our proofs, we are working on the formal proof of our theorems using the Coq proof assistant [1]. Coq provides an expressive language for defining not only mathematical objects but also datatypes and algorithms and for stating and proving their properties. The user builds proofs in Coq in an interactive manner. The formal proof of Theorem 4.1 and of the various properties that we have used to prove it in this paper is completed. We are now working on the formal proof of Theorem 4.2.

Acknowledgements We are extremely grateful to the anonymous referees, whose suggestions have been very helpful for revising this paper. Especially, one of them suggested a drastic simplification of the proof of Theorem 4.1.

References

1. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer (2004)
2. Boldo, S.: Pitfalls of a full floating-point proof: example on the formal proof of the Veltkamp/Dekker algorithms. In: U. Furbach, N. Shankar (eds.) Proceedings of the 3rd International Joint Conference on Automated Reasoning, *Lecture Notes in Computer Science*, vol. 4130, pp. 52–66 (2006)
3. Boldo, S., Daumas, M.: Representable correcting terms for possibly underflowing floating point operations. In: J.C. Bajard, M. Schulte (eds.) Proceedings of the 16th Symposium on Computer Arithmetic, pp. 79–86. IEEE Computer Society Press, Los Alamitos, CA (2003). DOI 10.1109/ARITH.2003.1207663
4. Boldo, S., Daumas, M., Moreau-Finot, C., Théry, L.: Computer validated proofs of a toolset for adaptable arithmetic. Tech. rep., École Normale Supérieure de Lyon (2001). Available at `http://arxiv.org/pdf/cs.MS/0107025`
5. Boldo, S., Melquiond, G.: Emulation of FMA and correctly rounded sums: proved algorithms using rounding to odd. IEEE Transactions on Computers **57**(4), 462–471 (2008)
6. Cornea, M., Harrison, J., Anderson, C., Tang, P.T.P., Schneider, E., Gvozdev, E.: A software implementation of the IEEE 754R decimal floating-point arithmetic using the binary encoding format. IEEE Transactions on Computers **58**(2), 148–162 (2009)
7. Dekker, T.J.: A floating-point technique for extending the available precision. Numerische Mathematik **18**(3), 224–242 (1971)
8. Figueroa, S.A.: When is double rounding innocuous? ACM SIGNUM Newsletter **30**(3) (1995)
9. Figueroa, S.A.: A rigorous framework for fully supporting the IEEE standard for floating-point arithmetic in high-level programming languages. Ph.D. thesis, Department of Computer Science, New York University (2000)
10. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys **23**(1), 5–47 (1991). An edited reprint is available at `http://www.physics.ohio-state.edu/~dws/grouplinks/floating_point_math.pdf` from Sun's Numerical Computation Guide; it contains an addendum *Differences Among IEEE 754 Implementations*, also available at `http://www.validlab.com/goldberg/addendum.html`.
11. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. SIAM, Philadelphia, PA (2002)
12. IEEE Computer Society: IEEE Standard for Floating-Point Arithmetic. IEEE Standard 754-2008 (2008). Available at `http://ieeexplore.ieee.org/servlet/opac?punumber=4610933`
13. International Organization for Standardization: Programming Languages – C. ISO/IEC Standard 9899:1999, Geneva, Switzerland (1999)
14. Kahan, W.: Pracniques: further remarks on reducing truncation errors. Commun. ACM **8**(1), 40 (1965). DOI 10.1145/363707.363723
15. Kahan, W.: Lecture notes on the status of IEEE-754 (1996). PDF file accessible at `http://www.cs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF`
16. Knuth, D.: The Art of Computer Programming, vol. 2, 3rd edn. Addison-Wesley, Reading, MA (1998)
17. Møller, O.: Quasi double-precision in floating-point addition. BIT **5**, 37–50 (1965)
18. Monniaux, D.: The pitfalls of verifying floating-point computations. ACM TOPLAS **30**(3), 1–41 (2008). A preliminary version is available at `http://hal.archives-ouvertes.fr/hal-00128124`
19. Muller, J.M., Brisebarre, N., de Dinechin, F., Jeannerod, C.P., Lefèvre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S.: Handbook of Floating-Point Arithmetic. Birkhäuser Boston (2010). ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9
20. Neumaier, A.: Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen. ZAMM **54**, 39–51 (1974). In German
21. Nievergelt, Y.: Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit. ACM Transactions on Mathematical Software **29**(1), 27–48 (2003)

22. Ogita, T., Rump, S.M., Oishi, S.: Accurate sum and dot product. SIAM Journal on Scientific Computing **26**(6), 1955–1988 (2005). DOI 10.1137/030601818
23. Pichat, M.: Correction d'une somme en arithmétique à virgule flottante. Numerische Mathematik **19**, 400–406 (1972). In French
24. Priest, D.M.: Algorithms for arbitrary precision floating point arithmetic. In: P. Kornerup, D.W. Matula (eds.) Proceedings of the 10th IEEE Symposium on Computer Arithmetic (Arith-10), pp. 132–144. IEEE Computer Society Press, Los Alamitos, CA (1991)
25. Priest, D.M.: On properties of floating-point arithmetics: Numerical stability and the cost of accurate computations. Ph.D. thesis, University of California at Berkeley (1992)
26. Rump, S.M., Ogita, T., Oishi, S.: Accurate floating-point summation part II: Sign, K-fold faithful and rounding to nearest. SIAM Journal on Scientific Computing (2005–2008). Submitted for publication
27. Rump, S.M., Ogita, T., Oishi, S.: Accurate floating-point summation part I: Faithful rounding. SIAM Journal on Scientific Computing **31**(1), 189–224 (2008). DOI 10.1137/050645671
28. Shewchuk, J.R.: Adaptive precision floating-point arithmetic and fast robust geometric predicates. Discrete Computational Geometry **18**, 305–363 (1997). URL `http://link.springer.de/link/service/journals/00454/papers97/18n3p305.pdf`
29. Sterbenz, P.H.: Floating-Point Computation. Prentice-Hall, Englewood Cliffs, NJ (1974)