# Necessary and sufficient conditions for exact floating point operations

Marc Daumas, Sylvie Boldo

# *Necessary and sufficient conditions for exact floating point operations*

Sylvie Boldo, LIP
Marc Daumas, CNRS

N° 4644

Novembre 2002

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4644--FR+ENG

# Necessary and sufficient conditions for exact floating point operations

Sylvie Boldo, LIP
Marc Daumas, CNRS

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arénaire

Rapport de recherche n° 4644 — Novembre 2002 — 18 pages

**Abstract:** Studying floating point arithmetic, authors have shown that the implemented operations (addition, subtraction, multiplication, division and square root) can compute a result and an exact correcting term using the same format as the inputs. Following a path initiated in 1965, all the authors supposed that neither underflow nor overflow occurred in the process. Overflow is not critical as some kind of exception is triggered by such an event that creates remanent non numeric quantities. Underflow may be fatal to the process as it returns wrong numeric values with little warning. Our new necessary and sufficient conditions guarantee that the exact floating point operations are correct when the result is a number. We also present properties when precise rounding is not available in hardware and faithful rounding alone is performed such as using some digital signal processing circuit. We have validated our proofs against the Coq automatic proof checker. Our development has raised many questions, some of them were expected while other ones were very surprising.

**Key-words:** Floating point, IEEE 754 Standard, Formal proof, Coq.

This text is also available as a research report of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP`.

# Conditions nécessaires et suffisantes pour un calcul à virgule flottante exact

**Résumé :** L'étude de l'arithmétique à virgule flottante a amené certains auteurs à démontrer que les opérations implantées (addition, soustraction, multiplication, division, racine carrée) peuvent calculer un résultat et un terme exact de correction en utilisant le même format que les entrées. Depuis 1965, tous les auteurs ont supposé qu'aucun dépassement de capacité vers l'infiniment petit ou vers l'infiniment grand ne se produisait. L'infiniment grand n'est pas dangereux car un évènement de ce type produit une exception associée à des quantités non numériques persistantes (NaN). L'infiniment petit peut être fatal au processus dans la mesure où il produit des résultat numériques faux avec peu d'avertissement. Nos nouvelles conditions nécessaires et suffisantes assurent que les opérations exactes à virgule flottante sont correctes quand le résultat est un nombre. Nous présentons aussi des résultats dans le cas où un arrondi précis n'est pas disponible en matériel et l'on effectue uniquement un arrondi fidèle comme c'est le cas lorsqu'on utilise certains circuits de traitement numérique du signal. Nous avons validé nos preuves grâce à l'assistant de preuve Coq. Notre développement a posé de nombreuses questions, nous nous attendions à certaines alors que d'autres nous ont surprises.

**Mots-clés :** Virgule flottante, Norme IEEE 754, Preuve formelle, Coq.

# Necessary and sufficient conditions for exact floating point operations

Sylvie Boldo & Marc Daumas
E-mail: Sylvie.Boldo@ENS-Lyon.Fr & Marc.Daumas@ENS-Lyon.Fr
Laboratoire de l'Informatique du Parallélisme
UMR 5668 CNRS – ENS de Lyon – INRIA
Lyon, France

## 1 Introduction

It was recognized in 1991 [3] that the most widely used algebraic operations of floating point arithmetic can return an exact correcting term provided the same floating point format is used for the inputs, the result and the correcting term. The first results on a similar subject were probably presented in 1965 [15, 23, 22] as techniques to enhance precision for additions and accumulations. Later work appeared in articles and textbooks [17, 11, 18].

As the IEEE 754 and 854 standards [30, 7] spread the use of correct rounding modes including rounding to the nearest floating point value, the rationale were studied for the four mentioned algebraic operations leading to a generic theorem such as the one below.

**Result 1 (Adapted from Bohlender *et al*, 1991)** *Let $\mathbb{F}$ be the set of real numbers represented with the defined floating point arithmetic and let $\oplus$, $\otimes$, $\oslash$, $\circ(\sqrt{\cdot})$ be the implemented addition, subtraction, multiplication, division and square root rounded to the nearest floating point value. Provided neither underflow nor overflow precludes a dependable behavior, an exact correcting term that is known to belong to $\mathbb{F}$ can be defined as follows from the inputs $x$ and $y$ and the result,*

| *Result* | *Correcting term* |
|---|---|
| $x \oplus y$ | $x + y - x \oplus y$ |
| $x \otimes y$ | $x \times y - x \otimes y$ |
| $x \oslash y$ | $x - (x \oslash y) \times y$ |
| $\circ(\sqrt{x})$ | $x - (\circ(\sqrt{x}))^2$ |

*The correcting term is still in $\mathbb{F}$ if we use a directed rounding mode for the multiplication $\otimes$ and the division $\oslash$.*

The fact that all the authors have overlooked the consequences of an overflow or an underflow in their work, may have arisen from an inadequacy of the existing formalisms to build a necessary and sufficient condition for an underflow to be harmless. We will see in this text that an error may occur even when neither inputs nor the result are subnormal (tiny) numbers.

A sufficient condition to be able to define an exact correcting term is that the exact error lies above the gradual underflow threshold as we will see in the conclusion of this text. This is not a necessary condition. We will present examples and counter examples in the text to show that all our conditions are both necessary and sufficient.

We introduce in the next section a new formalism where a number may have many different floating point notations. This formalism is used to model the numbers but not to compute on them. This is a very different use of redundant representations compared to common ones [2].

In the process of giving a tight necessary and sufficient condition for Result 1 to be correct even when underflow occurs, we have isolated a very surprising situation. In an intuitive deduction, the rounded result $r$ is the most significant part of the exact result and the correcting term $e$ is a remainder. It is easy to jump to a conclusion that $|e|$ must be significantly smaller than $|r|$ in additions, subtractions and multiplications, and smaller than $|x|$ in the divisions and the square roots. We have exhibited cases were $|e|$ is close to $|x|$. Exploring the directed rounding modes, we have found cases where $|e|$ is significantly larger than $|x|$.

All the theorems presented in this text have been developed with a strong focus on genericity. The radix, the number of significant digits of the mantissa, the underflow threshold and the rounding mode used are parameters of the theorems possibly set in the premises. Jumping to the conclusions of this text, we have no restriction on the radix provided it is an integer greater than or equal to 2 and no restriction on the number of digits of the mantissa provided once again it is an integer greater than or equal to 2. The tie breaking mechanism when rounding to the nearest has no effect on our theorems and can be even, odd or use any combination allowed in [25]. However, precise rounding to a nearest floating point number is necessary for the addition and the square root.

Following Sections 2 and 3, discussing our formalism and faithful rounding, we present the result on multiplications (Section 4), additions and subtractions (Section 5), divisions (Section 6), and square roots (Section 7) as difficulty increases. As our question is easily connected to the correct behavior of the remainder (FPREM) operation defined by the IEEE standard, we have built a machine-checked proof of this fact in the last section of this text, answering a question raised in 1995 [21] (Section 8). All the proofs can be downloaded through the Internet a the following address.

`http://www.ens-lyon.fr/~sboldo/coq/`

## 2 Floating point representation

Numbers are represented with pairs $(n, e)$ that stand for $n\beta^e$ where $\beta$ is the radix of the floating point system. In our characterization, we use both an integral signed mantissa $n$ and an integral signed amplitude $e$ for the sake of simplicity. The above definition is not sufficient to identify one unique pair $(n, e)$ for a represented quantity. We add a normalization convention whereby the $p$-digit magnitude of the mantissa of the **normal** representation of a number is required to start with a non zero digit. The underflow amplitude, a constant, is the lowest amplitude $-e_{\min}$ available.

We define a **bounded** pair $(n, e)$ such that $|n| < \beta^p$ and $e \geq -e_{\min}$. We do not set an upper bound on the amplitude as overflow are easily detected by other means. A bounded pair is normal if $\beta \cdot |n| \geq \beta^p$ and it is **subnormal** if $\beta \cdot |n| < \beta^p$ and $e = -e_{\min}$. Each represented number has one unique representation either normal or subnormal. A pair is **canonical** if it is either normal or subnormal.

In some sense, our internal representation is similar to the one proposed in the late 1950s [1, 32] and used again by one of the authors for a different development [9]. However, our approach is very different to the one proposed in this former work as the computer only manipulates the unique canonical representations. The other representations are just used to state necessary and sufficient conditions and prove the theorems.

In all the following theorems, we will use any representation of the floats. We do not have to use the canonical representation. Our theorems are valid when using the subnormals but no counter-example use them explicitly.

This formalism was introduced in [10] for our development using the Coq proof environment [13]. Other formalisms of the floating point arithmetic are in use with PVS [21, 14], HOL [6, 12] or ACL2 [27]. Using Curry Howard isomorphism, Coq and HOL rely on a very small inference engine to check the correctness of the proofs. Altough Coq and HOL lack many of the automatic techniques implemented in PVS or ACL, they allow the user to safely explore the properties of a system.

## 3 Faithful rounding

Most available general purpose processors have long been compliant with the IEEE 754 standard on floating point arithmetic [28]. It means that they implement precise rounding for the four arithmetic operations: addition, multiplication, division and square root. The result obtained for any of these computer operations is the one given by using a user-chosen rounding function on the result of the exact mathematical operation. The standard specifies four rounding functions: rounding to the nearest with the even tie breaking rule, rounding up, down or toward 0. The rounding functions and the arithmetic operators are defined and used in our formalism.

Precise rounding requires additional hardware to maintain a guard bit for subtraction and a sticky bit for all the operations [8]. Digital signal processing circuits that are designed for low cost and low power dissipation, such as Texas Instruments' TMS 320 C3x [31] may not implement precise rounding. Validating properties on such circuits becomes critical as these circuits are radiation-hardened for avionics and military developments

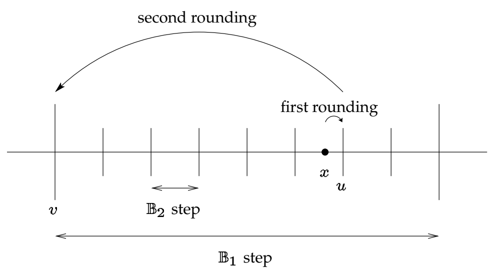

Figure 1: Double rounding of a real value

under the SMJ 320 C3x part number [5]. We model the behavior of such a circuit using the already defined notion of faithful rounding [9].

An operator implements faithful rounding if the result is either the rounded up or the rounded down value of the exact result. We assume that it is a non deterministic choice and the user has no ability to select the rounding mode *a priori* or know which rounding mode was used *a posteriori*. We have proved [4] that faithful rounding is the tightest non correct condition for floating point arithmetic (`MinOrMax_Rlt`, `MinOrMax1` and `MinOrMax2` in `Axpy.v`). It is more precise than allowing an error of one unit in the last place as the precision wobbles near the exact powers of the floating point radix.

Faithful rounding yields two more properties. First, we can use that a faithful rounded result is a non deterministic precise rounding. A property valid for any precise rounding mode is also valid when faithful rounding is applied and *vice versa*. Second, faithful rounding is stable through double rounding such as presented in the next theorem and Figure 1.

**Theorem 2 (DblRndStable in DoubleRound.v)** *Let $x$ be a real number and let $\mathbb{B}_1$ and $\mathbb{B}_2$ be two bounds such that any pair bounded by $\mathbb{B}_1$ can be represented with a pair bounded by $\mathbb{B}_2$. The double faithfull rounding of $x$ first with the fine bound $\mathbb{B}_2$ and second with the coarser bound $\mathbb{B}_1$ is also a faithfull rounding of $x$ with the coarse bound $\mathbb{B}_1$.*

In the theorem, $\mathbb{B}_2$ defines a subdivision of the discrete set defined by $\mathbb{B}_1$. For example, $\mathbb{B}_2$ can be associated with the double extended precision and $\mathbb{B}_1$ can be the bound of the IEEE double precision. It means that for $u$, any faithful rounding of $x$ in double extended precision, and $v$, any faithful rounding of $u$ in double precision, we know that $v$ is also a faithful rounding of $x$ in double precision.

On one hand, we will use the stability of faithful rounding to deduce that an exact correcting term can be computed for the multiplication and the division. On the other hand, we will present counter examples where strict precise rounding is necessary to represent an exact correcting term for the addition and the square root. Texas Instruments' circuit fails on these examples. The circuit also fails in providing the necessary and sufficient condition proposed in [24] to compute a bounded exact correcting term for the addition.

## 4 Multiplication

Concerning the question raised in this paper, the multiplication is most probably the simplest operation. The product of two $p$-digit mantissas produces a $2p$-digit mantissa that can be split into two $p$-digit floating point pairs whatever the rounding mode used.

This fact was recognized early and the IBM S/370 has a special instruction to produce these two pairs [16]. An algorithm has long been developed and tested to compute these pairs with only IEEE standard operations [11, 19] although it relies on some developments on the addition presented Section 5. The task of producing the two quantities is much simpler with a computer that provides a full accuracy fused multiply and accumulate operator such as with Intel IA64 [20].

The following theorem is based on some early development of our project.

**Theorem 3 (RepMult_gen in FroundMult)** *Let $\otimes$ be the implemented multiplication rounded to a nearest floating point value or with a directed rounding mode. Given inputs $x$ and $y$ such that $x \otimes y$ is neither an infinity nor a NaN, the correcting term*

$$x \times y - x \otimes y$$

*is bounded if and only if there exist two bounded pairs $(n_x, e_x)$ and $(n_y, e_y)$ representing $x$ and $y$ such that*

$$e_x + e_y \geq -e_{\min}.$$

**Counter example** We use the two following pairs representing the numbers $9 \times 2^{-\lfloor \frac{e_{\min}}{2} \rfloor -1}$ and $11 \times 2^{-\lceil \frac{e_{\min}}{2} \rceil}$. Our notation uses $p = 4$ digits and an arbitrary value for $e_{\min}$.

$$(1001_2, -\left\lfloor \frac{e_{\min}}{2} \right\rfloor - 1)_2 \quad \text{and} \quad (1011_2, -\left\lceil \frac{e_{\min}}{2} \right\rceil)_2$$

The exact product $99 \times 2^{-e_{\min}-1}$ rounds to the nearest floating point pair $96 \times 2^{-e_{\min}-1} = 12 \times 2^{-e_{\min}+2}$ represented by the pair $(1100_2, -e_{\min} + 2)_2$. The exact correcting term is $-3 \times 2^{-e_{\min}-1}$ and cannot be represented by a bounded pair. Still neither inputs nor the result is a subnormal pair.

## 5 Addition and subtraction

Authors have long exhibited two different situations in the production of an exact bounded correcting term for additions and subtractions. If the amplitudes of the inputs are close enough, the exact result can be written with a $2p$-digit mantissa. In this case, the rounded value and the error can be stored with a bounded pair whatever the rounding mode being either to the nearest or directed.

If the amplitudes of the inputs are too far away one from another, we have to make sure that the rounded result is the largest input in magnitude. This fact is obtained only when rounding to the nearest floating point value if the operations are precisely rounded. It was proved in some early part of our development [10] by adapting the proof of the correctness of the algorithm published in [29] to obtain the correcting term.

**Theorem 4 (errorBoundedPlus in ClosestPlus)** *Let $\oplus$ and $\ominus$ be the implemented addition and subtraction rounded to a nearest floating point value. Given inputs $x$ and $y$, for each result $x \oplus y$ and $x \ominus y$ that is neither an infinity nor a NaN, the correcting terms*

$$x + y - x \oplus y \quad \text{or} \quad x - y - x \ominus y$$

*are bounded.*

**Counter example** We present now an example where the double rounding of Texas Instruments' TMS 320 C3x introduces a non amendable error. Let the radix be $\beta = 2$ and the precision $p > 4$ arbitrary large. We assume that the extended precision less than doubles the number of digits in the mantissa. We compute the sum of the two normalized numbers $(-(2^{p-1} + 1), p + 1)_2$ and $(2^p - 3, 0)_2$.

The first input has value $-2^{2p} - 2^{p+1}$ and the exact result is $-2^{2p} - 2^p - 3$. As the extended precision is limited, the last two bits of the result are lost and the first rounding returns $-2^{2p} - 2^p$. This result is next rounded to a nearest by a truncation after adding a half unit in the last place. The result is thereafter $-2^{2p}$ and the error is $-2^p - 3$ that cannot be represented.

## 6 Division

Theorem 5 exhibits two necessary and sufficient conditions for the correcting term to be bounded. The first condition (1) is expected. The second condition (2) is very new and deals with a situation that only occurs with some of the directed rounding modes.

**Theorem 5 (errBoundedDiv in FroundDivSqrt.v)** *Let $\oslash$ be the implemented division rounded to a nearest floating point value or with a directed rounding mode. Given inputs $x$ and $y \neq 0$, whenever $x \oslash y$ is neither an infinity nor a NaN, the correcting term*

$$x - (x \oslash y) \times y$$

*is bounded if and only if there exists two bounded pairs $(n_y, e_y)$ and $(n_q, e_q)$ representing $y$ and $x \oslash y$ such that*

$$e_y + e_q \geq -e_{\min}, \tag{1}$$

*and*

$$|x \oslash y| \neq \beta^{-e_{min}} \quad \text{or} \quad \frac{\beta^{-e_{min}}}{2} \leq \left| \frac{x}{y} \right|. \tag{2}$$

We will prove that the exact remainder $r = x - (x \oslash y) \times y$ computed with appropriate care is bounded so that the two conditions are sufficient. We first define $q = x \oslash y$ with any rounding mode. We assume that there exists some representations $(n_q, e_q)$ of $q$ and $(n_y, e_y)$ of $y$ that satisfy (1) and let $(n_x, e_x)$ be a representation of $x$.

Let $(n'_q, e'_q)$ be the canonical representation of $q$, that means that

$$n_q \times \beta^{e_q} = n'_q \times \beta^{e'_q}$$

and we can define the **unit in the last place** function $\mathrm{ulp}(q) = \beta^{e'_q}$. We know from previous results that

$$e'_q \leq e_q \quad (\texttt{FcanonicLeastExp})$$

and

$$\left|\frac{x}{y} - q\right| < \mathrm{ulp}(q) \quad (\texttt{RoundedModeUlp}).$$

We will show that the floating point pair $(n_r, e_r)$ with

$$\begin{array}{rcl} n_r & = & n_x \times \beta^{e_x - \min(e_x, e_q + e_y)} \\ & - & n_q \times n_y \times \beta^{e_q + e_y - \min(e_x, e_q + e_y)} \\ e_r & = & \min(e_x, e_q + e_y) \end{array}$$

is a bounded representation of $r$. We easily check that $(n_r, e_r)$ is a representation of $x - qy$. To prove that $r$ is bounded, we consider two cases.

**First,** if $e_q + e_y \leq e_x$, then $e_r = e_q + e_y$ and $e_r \geq -e_{min}$ from (1). We check that

$$\begin{array}{rcl} |n_r| & = & |r| \times \beta^{-e_r} \\ & = & |x - qy| \times \beta^{-e_q - e_y} \\ & \leq & |y| \times \left|\frac{x}{y} - q\right| \times \beta^{-e_q - e_y} \\ & < & |n_y| \times \mathrm{ulp}(q) \times \beta^{-e_q} \\ & < & |n_y| \times \beta^{e'_q - e_q} \\ & < & |n_y| \end{array}$$

and finally, $|n_r| < \beta^p$.

**Second,** if $e_x < e_q + e_y$ then $e_r = e_x$ and $e_r \geq -e_{min}$. So $|n_r| < \beta^p$ is the only question left to finish our proof. We examine three cases depending on $|n'_q|$. If $|n'_q| = 0$, then $q = 0$ and $r = x$. If $|n'_q| > 1$, we check that

$$\begin{array}{rcl} \left|\frac{x}{y} - q\right| & < & \mathrm{ulp}(q) \\ |n'_q| \times |x - qy| & < & |n'_q| \times \mathrm{ulp}(q) \times |y| \\ & < & |q| \times |y| \\ & \leq & |x| + |x - qy| \end{array}$$

and finally $\left(|n'_q| - 1\right) \times |x - qy| \leq |x|$. Since $|n'_q| \geq 2$, we deduce that $|x - qy| \leq |x|$ and then $|n_r| \leq |n_x| < \beta^p$.

The last case lies when $|n'_q| = 1$. Since $q'$ is canonical $(n'_q, e'_q)$ is subnormal and $q = \pm\beta^{-e_{min}}$. We deduce $\beta^{-e_{min}}/2 \leq |x/y|$ from the second hypothesis (2) and $|x/y| < 2 \times \beta^{-e_{min}}$ since we used a rounding mode, so that the successor of the rounded result bounds the real value. We conclude that

$$\frac{|q|}{2} \leq \left|\frac{x}{y}\right| < 2\,|q| \quad \text{and} \quad \left|\left|\frac{x}{y}\right| - |q|\right| \leq \left|\frac{x}{y}\right|.$$

As $\frac{x}{y}$ and $q$ have the same sign (properties `RleRoundedR0` and `RleRoundedLessR0` of the rounding modes independent of the operation),

$$\left|\frac{x}{y} - q\right| \leq \left|\frac{x}{y}\right| \quad \text{and} \quad |x - qy| \leq |x|$$

That ends the proof.

**Counter examples** We will show that both hypotheses (1) & (2) are necessary and tight with an example when one of the hypotheses is not satisfied.

- The case where (1) is not satisfied follows an expected path. Let the radix be $\beta = 2$ and the precision $p = 4$ with

$$\begin{array}{rcl} x & = & 1001_2 \times 2^{-e_{min}+3} \\ y & = & 1101_2 \times 2^{-\lfloor \frac{e_{min}}{2} \rfloor}. \end{array}$$

  The division is rounded to the nearest, that is towards $-\infty$ here. We get

$$q = 1011_2 \times 2^{-1-\lceil \frac{e_{min}}{2} \rceil}.$$

  and $e_q + e_y = -e_{min} - 1$.
  The correcting term $x - qy$ is

$$r = -1101011_2 \times 2^{-e_{min}-1},$$

  that cannot be represented.

- The case where (2) is not satisfied is more surprising with

$$\begin{array}{rcl} x & = & \beta^{-e_{min}+p} \\ y & = & \beta^{2p+1}. \end{array}$$

  The division is rounded towards $+\infty$. We get

$$q = \beta^{-e_{min}}.$$

  The correcting term $x - qy$ is

$$-\left(\beta^{p+1} - 1\right) \times \beta^{-e_{min}+p}$$

  that cannot be represented and that is surprisingly much larger than $x$.

We have proved in theorems `errorBoundedDivClosest` and `errorBoundedDivToZero`, that hypothesis (2) is always true when rounding to a nearest floating point or towards zero.

We will discuss again in Section 8 on the choice of the method used to prove the theorems on the division. It would be difficult to use the one published in [3]. The proof is based on the usual division algorithm and properties that are not known *a priori* in a proof checking environment.

## 7 Square root

It is a common knowledge that square root extractions are similar to divisions in many ways. As stated in the following theorem only one condition (3) has to be satisfied. On the other hand, the correcting term can be defined only when the operator precisely rounds the result to a nearest floating point number.

**Theorem 6 (errBoundedSqrt in FroundDivSqrt.v)** *Let $\circ(\sqrt{\cdot})$ be the implemented square root operation rounded to a nearest floating point value. Given the input $x$, whenever $\circ(\sqrt{x})$ is neither an infinity nor a NaN, the correcting term*

$$x - \circ(\sqrt{x}) \times \circ(\sqrt{x})$$

*is bounded if and only if there exist a bounded pair $(n_q, e_q)$ representing $\circ(\sqrt{x})$ such that*

$$2e_q \geq -e_{\text{min}}. \tag{3}$$

Once again, we will prove that the exact remainder $r = x - \circ(\sqrt{x}) \times \circ(\sqrt{x})$ computed with appropriate care is bounded so that the condition is sufficient. We first define $q = \circ(\sqrt{x})$ rounded to the nearest floating point value. We assume that there exists some representations $(n_q, e_q)$ of $q$ that satisfies (3) and let $(n_x, e_x)$ be a representation of $x$.

Let $(n'_q, e'_q)$ be the canonical representation of $q$, that also means that

$$\left|\sqrt{x} - q\right| \leq \frac{\text{ulp}(q)}{2} \quad (\texttt{ClosestUlp}).$$

We will show that the floating point pair $(n_r, e_r)$ with

$$\begin{array}{rcl} n_r & = & n_x \times \beta^{e_x - \min(e_x, 2\, e_q)} \\ & - & n_q^2 \times \beta^{2\, e_q - \min(e_x, 2\, e_q)} \\ e_r & = & \min(e_x, 2\, e_q) \end{array}$$

is a bounded representation of $r$. We easily check that $(n_r, e_r)$ is a representation of $x - q^2$. To prove that $r$ is bounded, we consider two cases.

**First,** if $2\, e_q \leq e_x$, then $e_r = 2\, e_q$ and $e_r \geq -e_{min}$ from (3). We check that

$$\begin{array}{rcl} |n_r| & = & \left|x - q^2\right| \times \beta^{-2\, e_q} \\ & = & \left|\sqrt{x} - q\right| \times \left|\sqrt{x} + q\right| \times \beta^{-2\, e_q} \\ & \leq & \dfrac{\text{ulp}(q)}{2} \times \left(\left|\sqrt{x} - q\right| + 2\,|q|\right) \times \beta^{-2\, e_q} \\ & \leq & \dfrac{\text{ulp}(q)}{2} \times \left(\dfrac{\text{ulp}(q)}{2} + 2\,|q|\right) \times \beta^{-2\, e_q} \\ & \leq & \dfrac{1}{4} \times \beta^{2\, e'_q - 2\, e_q} + |n'_q| \times \beta^{2\, e'_q - 2\, e_q} \\ & \leq & \dfrac{1}{4} + |n'_q| \\ & \leq & \dfrac{1}{4} + (\beta^p - 1) \\ & \leq & \beta^p - \dfrac{3}{4} \end{array}$$

and finally, $|n_r| < \beta^p$.

**Second,** if $e_x < 2\, e_q$ then $e_r = e_x$ and $e_r \geq -e_{min}$. So $|n_r| < \beta^p$ is only left to finish our proof. We examine two cases depending on $(n'_q, e'_q)$ is normal or subnormal.

In the case where $(n'_q, e'_q)$ is normal, we use the fact that $\text{ulp}(q) \leq |q| \times \beta^{1-p}$ (`FulpLe2`) with $p \geq 2$, and that $q \geq 0$ (`RleRoundedR0`), so

$$\begin{array}{rcl} q & \leq & \sqrt{x} + \dfrac{\text{ulp}(q)}{2} \\ & \leq & \sqrt{x} + \dfrac{1}{2} \times \beta^{1-p} \times q \\ q & \leq & \dfrac{\sqrt{x}}{1 - \frac{\beta^{1-p}}{2}} \\ q^2 & \leq & x \times \left(\dfrac{1}{1 - \frac{\beta^{1-p}}{2}}\right)^2 \\ & \leq & 2\, x \end{array}$$

As $q^2$ and $x$ have the same sign and $q^2 \leq 2\, x$, we have $|x - q^2| \leq x$ and then again $|n_r| \leq |n_x| < \beta^p$.

This case where $(n'_q, e'_q)$ is subnormal cannot happen with any of the *common* single and double precision floating-point formats and it was expectedly dismissed by previous authors. But if the radix is $\beta = 2$, the precision is $p = 5$ and the underflow amplitude is $-e_{min} = -3$, the canonical representation of the square root of 1, the number represented by $(1, 0)$ is subnormal as it is $(1000_2, -3)$.

In this case, $e'_q = -e_{min}$ and

$$|n_r| \quad = \quad \left|x - q^2\right| \times \beta^{-e_x}$$

$$\begin{aligned}
&= \left|\sqrt{x}-q\right| \times \left|\sqrt{x}+q\right| \times \beta^{-e_x} \\
&\le \frac{\beta^{-e_{min}}}{2} \times \left(\left|\sqrt{x}-q\right| + 2\,\sqrt{x}\right) \times \beta^{-e_x} \\
&\le \frac{\beta^{-e_{min}}}{2} \times \left(\frac{\beta^{-e_{min}}}{2} + 2\,\sqrt{x}\right) \times \beta^{-e_x} \\
&\le \frac{1}{4}\beta^{-2\,e_{min}-e_x} + \sqrt{n_x}\sqrt{\beta^{e_x}}\beta^{-e_{min}-e_x} \\
&\le \frac{1}{4} + \sqrt{n_x} \times \sqrt{\beta^{-2e_{min}-e_x}} \\
&\le \frac{1}{4} + n_x \le \beta^p - \frac{3}{4}
\end{aligned}$$

and finally, $|n_r| < \beta^p$, so that the property still hold in this case never studied before.

**Counter examples** We will show once again that both hypotheses (3) and the rounding mode being to a nearest are necessary and tight with an example when one of the hypotheses is not satisfied.

- Let the radix be $\beta = 2$ and the precision $p = 4$. The case where (1) is not satisfied follows an expected path. We distinguish whether $e_{min}$ is even or not. In the first case, with

  $$\begin{aligned} x &= 1010_2 \times 2^{-e_{min}+1} \\ q &= 1001_2 \times 2^{-1-\frac{e_{min}}{2}}, \end{aligned}$$

  we check that $2\,e_q = -e_{min} - 2$ and the exact correcting term $x - q^2$ is

  $$-2^{-e_{min}-2},$$

  that cannot be represented in our floating-point format.

  If $e_{min}$ is odd (second case), we end up to the same conclusion with

  $$\begin{aligned} x &= 1010_2 \times 2^{-e_{min}+3} \\ q &= 1101_2 \times 2^{\frac{-1-e_{min}}{2}} \\ x - q^2 &= -1001_2 \times 2^{-e_{min}-1}. \end{aligned}$$

- To prove that precise rounding to a nearest is necessary for the square root operation, we focus on the number $x = 2^{2p+2} - 6 \times 2^{p+1}$ where the radix is $\beta = 2$ and the precision $p > 4$ is arbitrary large. The bounded representation of $x$ is $(2^p - 3, p+2)_2$.

  The square root of $x$ can be expressed in the form $\sqrt{x} = 2^{p+1} - 3 - \alpha \times 2^{-p}$ with $\alpha$ between $\frac{9}{16}$ and $\frac{9\sqrt{2}}{4}$ from Taylor-Lagrange inequality. Precise rounding would answer $2^{p+1} - 4$ represented by the pair $(2^p - 2, 2)_2$ but some hardware that discards $\alpha \times 2^{-p}$ may round the result to $2^{p+1} - 2$ and return the bounded pair $(2^p - 1, 2)_2$.

  In the later case, the correcting term $x - q^2$ is $-2 \times 2^{p+1} - 4 = -(2^p + 1) \times 4$ and it cannot be represented.

## 8 Remainder

The IEEE 754 and 854 standards define the remainder operation that shall be implemented for a system to comply to the standard. Given two inputs $x$ and $y$, we define $n$ that is the rounding of $x/y$ to the nearest integer value with the even tie breaking rule and the result is defined as

$$r = x - ny.$$

Both documents state that the remainder can always be represented using the same floating point format as the input. The best way to prove this assertion is to look at the common stepwise binary implementation of the Euclidean division. As the quotient is computed bit-wise, most significant bit first, the remainder always fit in the same format as the inputs. This invariant was also used in [3] to prove the theorem on divisions and the authors noticed that this invariant is not true for the stepwise square root extraction.

Porting this proof to an automatic proof checker is difficult as this would mean describing the Euclidean division with much details and properties. This is the reason why we have used a different technique and the question of the remainder being exact has never been answered with an automatic proof checker before [21]. The following theorem answers the question.

**Theorem 7 (errBoundedRem in FroundDivSqrt.v)** *Given inputs $x$ and $y \neq 0$, and $n$ the rounded value of $x/y$ to a nearest integer or with a directed rounding mode, the remainder*

$$x - n \times y$$

*is bounded.*

## 9 Conclusion

The urge for the development of automatic proof checking is evident from adventures of published proofs such as the ones described in [26]. We have proved and checked in this document old and new properties on floating point arithmetic. Some of the new properties are almost part of the common knowledge of the community of users of floating point arithmetic. However, validating them has made it possible to detect strange, very uncommon and counter-intuitive cases of failure. Should designers have decided to implement the functionality advocated in [3], such cases might have nurtured dormant bugs.

Although the properties shown in this document have been validated with an automatic proof checker, we do not regard them as unshakable truths. We have so far validated a large number of properties based on our specification and we have been able to connect many of these properties with results published in the literature. Working with an automatic proof checker, we like to compare it to a stubborn but helping colleague that will review our proof and will tirelessly ask for details of our proofs.

As a conclusion, we regard these properties as highly trusted. They incurred a significant amount of testing not reported in the process of the proof. Most proofs have first been written and approved as a pen an paper proof before being checked with the computer. The most uncommon achievement of this work is probably the ability to extend our highly trusted properties to digital signal processing circuits implementing a floating point arithmetic slightly different from the IEEE standard.

## 10 Acknowledgment

This work has benefited from the continuous help of Laurent Théry and Yves Bertot at the INRIA, in Sophia Antipolis. We have heavily used, modified and relied on the first specification of floating point arithmetic in Coq reported in [10] and partially funded by the AOC action of the INRIA.

The authors would also like to thank Peter Markstein for his help in promoting their work and his discussions on intermediate level properties of floating point arithmetic already in use in the development of mathematical libraries.

## References

[1] Robert L. Ashenhurst and Nicholas Metropolis. Unnormalized floating point arithmetic. *Journal of the ACM*, 6(3):415–428, 1959.

[2] Algirdas Avižienis. Signed digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, 10:389–400, 1961.

[3] Gerd Bohlender, Wolfgang Walter, Peter Kornerup, and David W. Matula. Semantics for exact floating point operations. In Peter Kornerup and David Matula, editors, *Proceedings of the 10th Symposium on Computer Arithmetic*, pages 22–26, Grenoble, France, 1991.

[4] Sylvie Boldo and Marc Daumas. Faithful rounding without fused multiply and accumulate. In *IMACS-GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, Paris, France, 2002.

[5] Sylvie Boldo and Marc Daumas. Properties of the subtraction valid for any floating point system. In *7th International Workshop on Formal Methods for Industrial Critical Systems*, pages 137–149, Málaga, Spain, 2002.

[6] Victor A. Carreño. Interpretation of IEEE-854 floating-point standard and definition in the HOL system. Technical Report Technical Memorandum 110189, NASA Langley Research Center, 1995.

[7] William J. Cody, Richard Karpinski, et al. A proposed radix and word-length independent standard for floating point arithmetic. *IEEE Micro*, 4(4):86–100, 1984.

[8] Jerome T. Coonen. Specification for a proposed standard for floating point arithmetic. Memorandum ERL M78/72, University of California, Berkeley, 1978.

[9] Marc Daumas and David W. Matula. Validated roundings of dot products by sticky accumulation. *IEEE Transactions on Computers*, 46(5):623–629, 1997.

[10] Marc Daumas, Laurence Rideau, and Laurent Théry. A generic library of floating-point numbers and its application to exact computing. In *14th International Conference on Theorem Proving in Higher Order Logics*, pages 169–184, Edinburgh, Scotland, 2001.

[11] Theodorus J. Dekker. A floating point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971.

[12] John Harrison. A machine-checked theory of floating point arithmetic. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors, *12th International Conference on Theorem Proving in Higher Order Logics*, pages 113–130, Nice, France, 1999.

[13] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. The Coq proof assistant: a tutorial: version 7.2. Technical Report 256, Institut National de Recherche en Informatique et en Automatique, Le Chesnay, France, 2002.

[14] Christian Jacobi. Formal verification of a theory of IEEE rounding. In *14th International Conference on Theorem Proving in Higher Order Logics*, pages 239–254, Edinburgh, Scotland, 2001. Supplemental Proceedings.

[15] William Kahan. Further remarks on reducing truncation errors. *Communications of the ACM*, 8(1):40, 1965.

[16] Alan H. Karp and Peter Markstein. High precision division and square root. *ACM Transactions on Mathematical Software*, 23(4):561–589, 1997.

[17] Donald E. Knuth. *The art of computer programming: Seminumerical Algorithms*. Addison Wesley, 1969. First edition.

[18] Seppo Linnainmaa. Analysis of some known methods of improving the accuracy of floating-point sums. *BIT*, 14:167–202, 1974.

[19] Seppo Linnainmaa. Software for doubled precision floating point computations. *ACM Transactions on Mathematical Software*, 7(3):272–283, 1981.

[20] Peter Markstein. *IA-64 and elementary functions: speed and precision*. Prentice Hall, 2000.

[21] Paul S. Miner. Defining the IEEE-854 floating-point standard in PVS. Technical Report Technical Memorandum 110167, NASA Langley Research Center, 1995.

[22] Ole Møller. Note on quasi double-precision. *BIT*, 5(4):251–255, 1965.

[23] Ole Møller. Quasi double-precision in floating point addition. *BIT*, 5(1):37–50, 1965.

[24] Michèle Pichat and Jean Vignes. *Ingénierie du contrôle de la précision des calculs sur ordinateur*. Editions Technip, 1993.

[25] John F. Reiser and Donald E. Knuth. Evading the drift in floating point addition. *Information Processing Letters*, 3(3):84–87, 1975.

[26] John Rushby and Friedrich von Henke. Formal verification of algorithms for critical systems. In *Proceedings of the Conference on Software for Critical Systems*, pages 1–15, New Orleans, Louisiana, 1991.

[27] David M. Russinoff. A mechanically checked proof of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7 processor. *LMS Journal of Computation and Mathematics*, 1:148–200, 1998.

[28] N. L. Schryer. A test of computer's floating-point arithmetic unit. Technical report 89, AT&T Bell Laboratories, 1981.

[29] Jonathan R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. In *Discrete and Computational Geometry*, volume 18, pages 305–363, 1997.

[30] David Stevenson et al. An american national standard: IEEE standard for binary floating point arithmetic. *ACM SIGPLAN Notices*, 22(2):9–25, 1987.

[31] Texas Instruments. *TMS320C3x — User's guide*, 1997.

[32] W. G. Wadey. Floating point arithmetics. *Journal of the ACM*, 7(2):129–139, 1960.

## Module FroundDivSqrt

Require Export *ClosestProp*.
Require *Import Classical*.
Section *FroundDiv*.
Variable *b* : *Fbound*.
Variable *radix* : *Z*.
Variable *precision* : *nat*.

*Coercion* Local *FtoRradix* := (*FtoR radix*).
Hypothesis *radixMoreThanOne* : (*Zlt* (*POS xH*) *radix*).

Local *radixMoreThanZERO* := (*Zlt_1_O* ? (*Zlt_le_weak* ? ? *radixMoreThanOne*)).
Hints *Resolve radixMoreThanZERO* :*zarith*.
Hypothesis *precisionGreaterThanOne* : (*lt* (*S O*) *precision*).
Hypothesis *pGivesBound* : (*POS* (*vNum b*)) = (*Zpower_nat radix precision*).

Theorem *FulpFabs*:
 (*f* : *float*)
  <*R*> (*Fulp b radix precision f*) == (*Fulp b radix precision* (*Fabs f*)).

Theorem *NearestInteger*:
 (*r* : *R*)
 (*n* : *Z*)
 ((*z* : *Z*) (*Rle* (*Rabsolu* (*Rminus n r*)) (*Rabsolu* (*Rminus z r*)))) →
  (*Rle* (*Rabsolu* (*Rminus n r*)) (*Rinv* (*Rplus R1 R1*))).

Theorem *errorBoundedModulo_aux*:
 (*x*, *y* : *float*)
 (*n* : *Z*)
 (*Fbounded b x*) →
 (*Fcanonic radix b x*) →
 (*Fbounded b y*) →
 (*Fcanonic radix b y*) →
 ¬ <*R*> *y* == *R0* →
 (*Rlt R0 y*) →
 (*Rle R0 x*) →
 ((*z* : *Z*) (*Rle* (*Rabsolu* (*Rminus n* (*Rdiv x y*))) (*Rabsolu* (*Rminus z* (*Rdiv x y*))))) →
  (*Fbounded b* (*Fminus radix x* (*Fmult* (*Float n ZERO*) *y*))).

Theorem *errorBoundedModulo_aux_y*:
 (*x*, *y* : *float*)
 (*n* : *Z*)
 (*Fbounded b x*) →
 (*Fcanonic radix b x*) →
 (*Fbounded b y*) →
 (*Fcanonic radix b y*) →
 ¬ <*R*> *y* == *R0* →
 (*Rle R0 x*) →
 ((*z* : *Z*) (*Rle* (*Rabsolu* (*Rminus n* (*Rdiv x y*))) (*Rabsolu* (*Rminus z* (*Rdiv x y*))))) →
  (*Fbounded b* (*Fminus radix x* (*Fmult* (*Float n ZERO*) *y*))).

Theorem *errorBoundedModuloCan*:
 (*x*, *y* : *float*)
 (*n* : *Z*)
 (*Fbounded b x*) →
 (*Fcanonic radix b x*) →

```
 (Fbounded b y) →
 (Fcanonic radix b y) →
 ¬ <R> y == R0 →
 ((z : Z) (Rle (Rabsolu (Rminus n (Rdiv x y))) (Rabsolu (Rminus z (Rdiv x y))))) →
  (Fbounded b (Fminus radix x (Fmult (Float n ZERO) y))).

Theorem errBoundedRem:
 (x, y : float)
 (n : Z)
 (Fbounded b x) →
 (Fbounded b y) →
 ¬ <R> y == R0 →
 ((z : Z) (Rle (Rabsolu (Rminus n (Rdiv x y))) (Rabsolu (Rminus z (Rdiv x y))))) →
  (Fbounded
   b
   (Fminus
    radix (Fnormalize radix b precision x)
    (Fmult (Float n ZERO) (Fnormalize radix b precision y)))).

Theorem errBoundedDiv:
 (x, y, q : float)
 (P : ?)
 (RoundedModeP b radix P) →
 (Fbounded b x) →
 (Fbounded b y) →
 (Fbounded b q) →
 ¬ <R> y == R0 →
 (P (Rdiv x y) q) →
 (Zle (Zopp (dExp b)) (Zplus (Fexp q) (Fexp y))) →
 ¬ <R> (Rabsolu q) == (powerRZ radix (Zopp (dExp b))) ∨
 (Rle (Rdiv (powerRZ radix (Zopp (dExp b))) (S (S O))) (Rabsolu (Rdiv x y))) →
  (Fbounded b (Fminus radix x (Fmult q y))).

Theorem errorBoundedDivSimplHyp:
 (x, y, q : float)
 (P : ?)
 (RoundedModeP b radix P) →
 (Fbounded b x) →
 (Fbounded b y) →
 (Fbounded b q) →
 ¬ <R> y == R0 →
 (P (Rdiv x y) q) →
 (Zle
  (Zopp (dExp b)) (Zminus (Fexp (Fnormalize radix b precision x)) precision)) →
  (Zle (Zopp (dExp b)) (Zplus (Fexp q) (Fexp y))).

Theorem errorBoundedDivClosest:
 (x, y, q : float)
 (Fbounded b x) →
 (Fbounded b y) →
 (Fbounded b q) →
 ¬ <R> y == R0 →
 (Closest b radix (Rdiv x y) q) →
 (Zle (Zopp (dExp b)) (Zplus (Fexp q) (Fexp y))) →
  (Fbounded b (Fminus radix x (Fmult q y))).

Theorem errorBoundedDivToZero:
 (x, y, q : float)
 (Fbounded b x) →
 (Fbounded b y) →
 (Fbounded b q) →
 ¬ <R> y == R0 →
 (ToZeroP b radix (Rdiv x y) q) →
 (Zle (Zopp (dExp b)) (Zplus (Fexp q) (Fexp y))) →
   (Fbounded b (Fminus radix x (Fmult q y))).

Theorem errBoundedSqrt:
 (x, q : float)
 (Fbounded b x) →
 (Fbounded b q) →
 (Rle R0 x) →
 (Closest b radix (sqrt x) q) →
 (Zle (Zopp (dExp b)) (Zplus (Fexp q) (Fexp q))) →
   (Fbounded b (Fminus radix x (Fmult q q))).
End FroundDiv.
```

## Module DoubleRound

```
Require Export AllFloat.
Section MOMR.
Variables b1, b2 : Fbound.
Variable radix : Z.
Variables prec1, prec2 : nat.

Coercion Local FtoRradix := (FtoR radix).
Hypothesis radixMoreThanOne : (Zlt (POS xH) radix).
Hypothesis prec1GreaterThanOne : (lt (S O) prec1).
Hypothesis prec2GreaterThanOne : (lt (S O) prec2).
Hypothesis p1GivesBound : (POS (vNum b1)) = (Zpower_nat radix prec1).
Hypothesis p2GivesBound : (POS (vNum b2)) = (Zpower_nat radix prec2).

Definition MinOrMax :=
    [b : Fbound] [z : R] [f : float] (isMin b radix z f) ∨ (isMax b radix z f).

Theorem MinOrMax_Rlt:
  (b : Fbound)
  (precision : nat)
  (z : R)
  (p : float)
  (lt (S O) precision) →
  (POS (vNum b)) = (Zpower_nat radix precision) →
  (MinOrMax b z p) → (Rlt (Rabsolu (Rminus z p)) (Fulp b radix precision p)).

Theorem BoundedBounded:
  (f : float)
  (le prec2 prec1) →
  (Zminus prec1 prec2) == (Zminus (dExp b1) (dExp b2)) →
  (Fbounded b2 f) → (Fbounded b1 f).

Theorem DblRndStable:
  (z : R)
  (p : float)
  (q : float)
  (le prec2 prec1) →
  (Zminus prec1 prec2) == (Zminus (dExp b1) (dExp b2)) →
  (Fbounded b1 p) →
  (Fbounded b2 q) → (MinOrMax b1 z p) → (MinOrMax b2 p q) → (MinOrMax b2 z q).

Theorem DoubleRound2:
  (z : R)
  (p : float)
  (q : float)
  (le prec2 prec1) →
  (Zminus prec1 prec2) == (Zminus (dExp b1) (dExp b2)) →
  (Fbounded b1 p) →
  (Fbounded b2 q) →
  (MinOrMax b1 z p) →
  (Closest b2 radix p q) →
    (Rlt
      (Rabsolu (Rminus z q))
      (Rmult
        (Fulp b2 radix prec2 q)
        (Rplus (Rinv (S (S O))) (powerRZ radix (Zs (Zminus prec2 prec1)))))).
```

End *MOMR*.

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399