A Formal Proof of Demmel and Hida's Accurate Summation Algorithm

Laurent Fousse [a], Paul Zimmermann [a]

[a] *LORIA/INRIA Lorraine, 615 rue du jardin botanique, F-54602 Villers-lès-Nancy Cedex*

Abstract

A new proof of the "accurate summation" algorithm proposed by Demmel and Hida is presented. The main part of that proof has been written in the Coq language and verified by the Coq proof assistant.

Key words: Floating-point arithmetic, bounded error, formal proof.

---

1 Motivation and Introduction

Computer arithmetic — especially floating-point arithmetic — is a challenging domain for formal proofs and the use of proof assistants. One reason is that only few proof assistants have specialized libraries for numerical computations. In addition, floating-point computations usually require dealing with inequalities, due to rounding.

After the "Pentium bug" in 1994, hardware manufacturers started to use proof assistants to develop new processors. First results were obtained by Harrison, who formally verified using HOL Light the correctness of the division, square root, and transcendental functions of Intel's Itanium, and Russinoff who proved using ACL2 the division and square root of AMD's K7 [8]. However, using a proof assistant is still far from being common in the computer arithmetic community. As an example, no formalization is known for the IEEE 754 standard [5].

Email addresses: `laurent@fousse.info` (Laurent Fousse), `zimmerma@loria.fr` (Paul Zimmermann).

In July 2002, Daumas gave a talk to the revision committee of IEEE 754, in order to advertize formal specifications and proofs. Kahan's answer was:

> *"I approve of what you're doing. I regard it as necessary. My hope is that continuing in that direction will ultimately lead to specifications and implementations which are not merely verifiable, but also maintainable."*

and the minutes from the meeting said: "*Kahan and Daumas also briefly debated applicability of automated verification techniques to simulated extra precision. Kahan thought Demmel and Hida's paper on accurate summations might provide difficult examples.*" This was a very good motivation to get a formal proof of Demmel and Hida's "accurate summation" algorithm [3].

Adding several floating-point numbers with good accuracy is an important problem of scientific computing. The best possible result is to provide *correct rounding* within the target precision and with respect to the given rounding mode. For a sum of two numbers, this can be obtained with a configuration following the IEEE 754 standard. For a sum of three or more numbers, not only the floating-point addition is not associative, but some cancellations may occur in the intermediate sums, that will make the final result completely wrong. Several algorithms have been proposed to overcome this problem. Floating-point expansions [7] enable one to emulate an arbitrary-precision arithmetic using fixed-precision floating-point numbers: an intermediate result is represented by a sum of disjoint floating-point numbers, so that the most significant one represents the rounding to nearest of the result.

We focus in this paper on algorithms that use an internal register, with a precision that is larger than that of the summands and the final result (which we assume to have same precision for simplicity). Kulish and Bohlender propose in [6] an algorithm that provides correct rounding of a dot product of n-digit vectors, with an accumulator of $2n + 2$ digits plus two bits. However, Kulish-Bohlender's algorithm is quite expensive to implement, since it requires to sort the inputs by decreasing exponent, and to combine those of same exponent (in a way similar to floating-point expansions).

More recently, Demmel and Hida proposed in [2,3] a simple algorithm that yields a sum correct to within about 1.5 ulps, as long as the number of summands is not too large. The proof of the main theorem from [3] is quite complex; quoting [2]: "*it* [the proof of Theorem 1] *involves a detailed case analysis requiring* 19 *pages*". Such a long proof — with 6 cases and 11 subcases — is difficult to completely grasp by a human-being, and is therefore a good candidate for the use of a proof assistant.

We provide in this paper a formal proof of the main case of Theorem 1 from [3], with the help of the Coq proof assistant [9]. We had to build a completely new proof, since the original one contained statements that were difficult to put in computer form, for example:

Fact 1. *The rightmost nonzero bit of $\widehat{SUM}_k$ must lie at or to the left of the trailing bit of s_k. This is because $\widehat{SUM}_k$ is gotten by summing s_1 through s_k, whose trailing bits are at or to the left of the trailing bit of s_k, since the exponents of s_1 through s_k are in decreasing order.*

We also give examples showing that the bound in that main case are optimal (Demmel and Hida examples did not cover that case).

Our paper is organized as follows. Section 2 introduces the notations and assumptions used in the rest of the paper, recalls Demmel and Hida's algorithm and states the main theorem we prove. Section 3 provides a new proof for the error bound in the main case we consider. Section 4 gives examples reaching that bound, thus showing that it is optimal. Section 5 expresses in human-readable form the main result (Lemma 5) that was actually checked using the Coq system, with some final remarks in Section 6.

2 Demmel and Hida's Algorithm

We consider here *binary* floating-point numbers only.

2.1 Notations

Exponent. For x a non-zero real number (not necessarily a floating-point number), we define $E(x) := 1 + \lfloor \log_2 |x| \rfloor$, such that $2^{E(x)-1} \leq |x| < 2^{E(x)}$.

Unit in last place. For x a non-zero real number (not necessarily a floating-point number), and an integer $f \geq 1$, we define $\mathrm{ulp}_f(x) := 2^{E(x)-f}$. When x is a floating-point number with a mantissa of f bits, $\mathrm{ulp}_f(x)$ corresponds to the weight of the last mantissa bit.

Rounding. We consider here only rounding to nearest. We denote $\circ_F(x)$ the F-bit floating-point number that is nearest from the real x. When the rounding precision is implicit, we simply write $\circ(x)$. Except explicitely stated, we don't require a functional definition for rounding; we only use the fact that the rounding error is bounded by $\frac{1}{2}\mathrm{ulp}(\circ_F(x))$.

Exact/Inexact. We say that a computation $\circ(x+y)$ is *exact* (resp. *inexact*) when the result is (resp. is not) exactly $x+y$. When that result is stored in a variable, like $z \leftarrow \circ(x+y)$, we say that "z is exact", meaning that $z = x + y$.

In the following we consider floating-point numbers with *unbounded exponent*, i.e. no underflow nor overflow occurs.

2.2 Demmel and Hida's algorithm

Consider the following sequence of operations, where all s_i and S are floating-point numbers with precision f, the t_i have precision $F > f$, and the s_i have non-increasing exponents (we assume that zero summands were removed before):

Algorithm DemmelHida.
$t_0 \leftarrow 0$
for $i \leftarrow 1$ **to** n
  $t_i \leftarrow \circ_F(t_{i-1} + s_i)$
$S \leftarrow \circ_f(t_n)$

Extending the above notations, we will say "the summation is inexact" when at least one rounding is inexact (although the final sum S returned by the algorithm may equal the true sum).

Demmel and Hida give in [3, Theorem 1] several bounds for the final error on S, depending on how large the number n of summands is, compared to the bound

$$\bar{n} = 1 + \left\lfloor \frac{2^F}{2^f - 1} \right\rfloor .$$

The main case is $n \leq \bar{n}$, where Demmel and Hida prove that the error on S is bounded by $\frac{1}{1-2^{1-f}} + \frac{1}{2}$ ulps.

Note that for $F < 2f$ the bound $\bar{n}$ simplifies to $2^{F-f} + 1$. We focus on that simpler bound here:

Theorem 1 *For $F > f \geq 2$, rounding to nearest, and $n \leq 2^{F-f} + 1$, the final error on S in Algorithm DemmelHida is bounded by* 1.5 *ulps.*

In addition to Demmel and Hida notations, we define $k := F - f$.

3 Bounds

3.1 Notations

Error and total error. Let $\varepsilon_j := |t_j - (t_{j-1} + s_j)|$ be the absolute rounding error on t_j — we will simply say the "error on t_j" —, and $\varepsilon_{i,j} := \sum_{i<l\le j} \varepsilon_l$ the cumulated rounding error when summing s_{i+1} to s_j. The final error on t_n is bounded by $\varepsilon_{0,n}$. (Please note that since some rounding errors may cancel each other, the values of the $\varepsilon_{i,j}$ are always a pessimistic majoration.)

First error index. We define i such that t_i is the first inexact partial sum, i.e. $t_i \neq t_{i-1} + s_i$. If such a i does not exist — which means that all partial sums are exact — we set by convention $i = n + 1$.

Exponent difference. For $1 \le i \le j \le n$, we define $e_j := E(t_j) - E(t_i)$ to be the exponent difference between t_j and t_i.

Definition 1 *In the case $i \le n$ — i.e. some rounding error occurs during the summation — we define i' to be the largest index such that $E(t_{i'}) \ge E(t_i)$ (Fig. 1).*

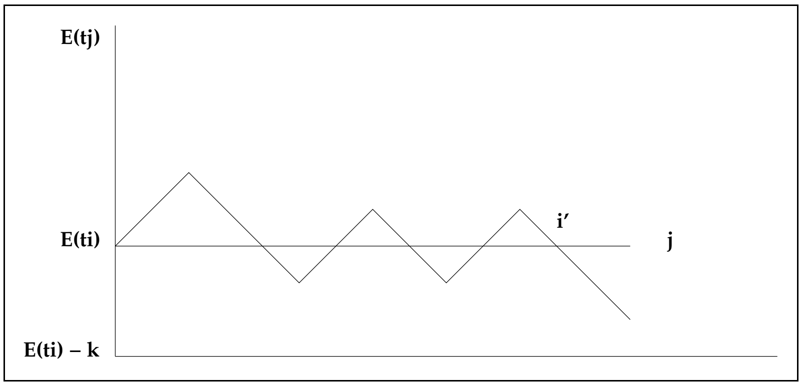

Fig. 1. Graph of $E(t_j)$ with respect to j, for $1 \le j \le n$, with special values i for first inexact result, and i' for last exponent $\ge E(t_i)$.

We clearly have $i' \ge i$. The total (rounding) error from summands s_j for $j < i$ is zero by definition of i, and we will distinguish the error for $i \le j \le i'$, and that for $i' < j \le n$ (Lemma 5).

We prove here a few lemmas that will help us to prove Theorem 1. The first result is that after the first inexact partial sum t_i, the summands s_j cannot be too large (Corollary 1), and thus the partial sums t_j cannot decrease too rapidly (Lemma 4). The key idea of the proof is that when the exponent decreases by one from t_{j-1} to t_j, the total error accumulated so far is multiplied by two, since it is now expressed in terms of $\mathrm{ulp}_F(t_j)$, but on the other side the maximum absolute value of subsequent summands s_l for $l > j$ that may produce a rounding error is divided by two. Lemma 5 expresses this trade-off between having large s_j (and thus making t_j decrease rapidly) and accumulating rounding errors.

Lemma 1 *If* $\mathrm{ulp}_f(s_j) \geq \mathrm{ulp}_F(t_j)$*, then* t_j *is exact.*

PROOF. By definition of rounding to nearest, the rounding error on t_j is at most $\frac{1}{2}\mathrm{ulp}_F(t_j)$. Since t_{j-1} is the (rounded) sum of s_1 to s_{j-1}, which have an exponent greater or equal to that of s_j, t_{j-1} is an integer multiple of $\mathrm{ulp}_f(s_j)$. Then $t_{j-1} + s_j$ is also an integer multiple of $\mathrm{ulp}_f(s_j)$, thus of $\mathrm{ulp}_F(t_j)$ since by hypothesis $\mathrm{ulp}_f(s_j) \geq \mathrm{ulp}_F(t_j)$. The only one such multiple at distance $\leq \frac{1}{2}\mathrm{ulp}_F(t_j)$ of t_j being t_j itself, the latter is exact. $\square$

REMARK. The fact that t_{j-1} is the sum of numbers with exponent larger or equal to that of s_j is crucial here: the above result is wrong for $t' := \circ(t+s)$. Take for example $f = 2$, $F = 3$, $t = 0.111$, $s = 0.11$, then $t' = \circ_3(1.101) = 1.10$ is inexact, while $\mathrm{ulp}(s) \geq \mathrm{ulp}(t')$ holds. (However, a sufficient auxiliary condition is $\mathrm{ulp}(t') \leq \mathrm{ulp}(t)$.)

It follows from Lemma 1 applied for $j = i$, and because the exponents are non-increasing:

Corollary 1 *If the summation is inexact, then* $\mathrm{ulp}_f(s_j) < \mathrm{ulp}_F(t_i)$ *for* $j \geq i$*.*

We will use several times this Corollary, mainly to bound $|s_j|$; indeed, since $\mathrm{ulp}_f(s_j) \leq \mathrm{ulp}_F(t_i) - 1$, and s_j has precision f, it follows $|s_j| < 2^{f-1}\mathrm{ulp}_F(t_i)$.

Lemma 2 *If* $i' < n$*, then* $E(t_{i'}) = E(t_i)$*.*

PROOF. From Def. 1, we have $E(t_{i'}) \geq E(t_i)$. Assume $E(t_{i'}) > E(t_i)$. Then $|t_{i'}| \geq 2^{F-1}\mathrm{ulp}_F(t_{i'}) \geq 2^F\mathrm{ulp}_F(t_i)$. Thus using Cor. 1 $|t_{i'} + s_{i'+1}| \geq (2^F - 2^{f-1})\mathrm{ulp}_F(t_i) \geq 2^{F-1}\mathrm{ulp}_F(t_i)$. It follows $E(t_{i'+1}) \geq E(t_i)$, which contradicts Def. 1. $\square$

Lemma 3 *Let* t *be a* F*-bit floating-point number, and* s *a real number. If* $|s| < 2^l$ *for* l *integer,* $E(t) - F \leq l < E(t)$*, and* $t' = \circ_F(t+s)$*, then* $|t' - t| \leq 2^l$*.*

PROOF. If $t = 0$, then $|t' - t| = |\circ_F(s)| \leq 2^l$ since 2^l is exactly representable. Without loss of generality we can consider that t is positive (remember rounding to nearest is symmetric with respect to 0).

First consider s negative. The condition $E(t) - F \leq l < E(t)$ implies $1 \leq \frac{2^l}{\mathrm{ulp}(t)} \leq 2^{F-1}$. Since t can be written $m \cdot \mathrm{ulp}(t)$ with $2^{F-1} \leq m < 2^F$, we deduce $0 \leq \frac{t-2^l}{\mathrm{ulp}(t)} < 2^F$, thus $t - 2^l$ — which is an integer multiple of $\mathrm{ulp}(t)$ — is exactly representable with a precision of F bits. The real $t+s$ is enclosed by two representable numbers $t - 2^l$ and t, thus the same holds for its rounding:

$t - 2^l \leq t' \leq t$, and the statement of the lemma follows.

Now consider s positive. If $t + 2^l$ is exactly representable, the same reasoning as above holds. If $t + 2^l$ is not representable on F bits, then it is the middle of two consecutive F-bit representable numbers $u < u^+$. Since $t + s < t + 2^l$, necessarily $t' := \circ(t + s) \leq u < t + 2^l$. □

Corollary 2 *With the notations of Lemma 3, if s is a f-bit floating-point number, and* $\mathrm{ulp}_f(s) \leq \mathrm{ulp}_F(t')$, *then* $|t'| \geq |t| - (2^f - 1)\mathrm{ulp}_F(t')$.

Proof. Assume t positive. If s is positive, the corollary is trivial. If $\mathrm{ulp}_f(s) < \mathrm{ulp}_F(t')$, Lemma 3 gives $|t' - t| \leq 2^{f-1}\mathrm{ulp}_F(t')$. If $\mathrm{ulp}_f(s) = \mathrm{ulp}_F(t')$, t' is exact by Lemma 1, thus $|t'| \geq |t| - |s| \geq |t| - (2^f - 1)\mathrm{ulp}_F(t')$. □

Lemma 4 *Assume $i \leq n$, and let i' as in Def. 1. Then for $i' < j < i' + 2^k$, we have $E(t_j) \geq E(t_i) - k$.*

Proof. (Remember that $k := F - f$.) Consider $j > i'$. Corollary 1 gives $\mathrm{ulp}_f(s_j) < \mathrm{ulp}_F(t_i)$. Thus from Corollary 2 the maximum decrease due to s_j is $\frac{1}{2}(2^f - 1)\mathrm{ulp}_F(t_i)$. When summing $2^k - 1$ terms for $i' < j < i' + 2^k$, we have:

$$\begin{aligned} |t_j| &\geq |t_{i'}| - \tfrac{1}{2}(2^k - 1)(2^f - 1)\mathrm{ulp}_F(t_i) \\ &\geq \tfrac{1}{2}[2^F - (2^k - 1)(2^f - 1)]\mathrm{ulp}_F(t_i) \geq 2^{f-1}\mathrm{ulp}_F(t_i), \end{aligned}$$

since $2^f - 1 \geq 2^{f-1}$. □

Lemma 5 *For $i' < j \leq n$ the cumulated error bound $\varepsilon_{i',j}$ when summing $s_{i'+1}$ to s_j satisfies:*

$$2^f \varepsilon_{i',j} \leq |t_j| - |t_{i'}| + (j - i')2^{f-1}\mathrm{ulp}_F(t_{i'}).$$

Proof. See Section 5 for the case $i' < n$. (For $i' = n$ there is nothing to prove.)

From $e_j = E(t_j) - E(t_i)$ and Lemma 2, we have $\mathrm{ulp}_F(t_{i'}) = 2^{-e_j}\mathrm{ulp}_F(t_j)$, thus Lemma 5 gives:

$$\varepsilon_{i',j} \leq [2^k - 2^{k-e_j-1} + (j - i')2^{-e_j-1}]\mathrm{ulp}_F(t_j). \tag{1}$$

Theorem 2 *Assume the summation is inexact. Then for $n \leq i + 2^k - 1$, the total error on t_n satisfies $\varepsilon_{0,n} < 2^k\mathrm{ulp}_F(t_n)$.*

Proof. First assume that $E(t_j) \leq E(t_i)$ for $j \geq i$. Let i' as in Def. 1. The maximal error when summing s_i up to $s_{i'}$ (both included) is $\frac{i'-i+1}{2}\mathrm{ulp}_F(t_i) = (i' - i + 1)2^{-e_n-1}\mathrm{ulp}_F(t_n)$. Together with the bound on the cumulated error when adding $s_{i'+1}$ to s_n given by Eq. (1), this gives a total error on t_n less than:

$$[2^k + 2^{-e_n-1}(n-i+1-2^k)]\mathrm{ulp}_F(t_n).$$

For $n \leq i + 2^k - 1$, that error is bounded by $2^k\mathrm{ulp}_F(t_n)$, and the theorem follows.

Consider now that there exists $l > i$ such that $E(t_l) > E(t_i)$. We firstly prove that

$$E(t_i) \leq E(t_j) \leq E(t_i) + 1 \tag{2}$$

in that case. For $i < j \leq n$, $|t_j| \geq |t_l| - (n-i+1)2^{f-1}\mathrm{ulp}_F(t_i) \geq [2^F - 2^{k+f-1}]\mathrm{ulp}_F(t_i) = 2^{F-1}\mathrm{ulp}_F(t_i)$. Thus $E(t_j) \geq E(t_i)$. On the other side, $|t_j| \leq |t_i| + (n-i+1)2^{f-1}\mathrm{ulp}_F(t_i) < 2^F[1+\frac{1}{2}]\mathrm{ulp}_F(t_i) \leq 2^{F+1}\mathrm{ulp}_F(t_i)$. It follows $E(t_j) \leq E(t_i)+1$, which completes the proof of (2). The total error on t_n satisfies:

$$\varepsilon_{0,n} < \frac{1}{2}\sum_{j=i}^{n} 2^{E(t_j)-E(t_n)}\mathrm{ulp}_F(t_n) \leq (n-i+1)\mathrm{ulp}_F(t_n) \leq 2^k\mathrm{ulp}_F(t_n). \qquad \square$$

Since $i \geq 2$ (t_1 is always exact), we get the following corollary.

Corollary 3 *For $n \leq 2^k + 1$, the total error on t_n satisfies $\varepsilon_{0,n} < 2^k\mathrm{ulp}_F(t_n)$.*

This completes the proof of Theorem 1 when the summation is inexact: that bound of $2^k\mathrm{ulp}_F(t_n)$ converts to $1 \cdot \mathrm{ulp}_f(S)$, to which we add the final rounding error of at most $\frac{1}{2}\mathrm{ulp}_f(S)$ when rounding t_n back to f bits.

4 Reaching the Bounds

We assume here even rounding, with ties broken by rounding to the adjacent floating-point number with a mantissa ending by a zero.[1]

Lemma 6 *When summing $n = 1 + 2^k$ terms, we can get as near as desired to the error bound of $1.5\mathrm{ulp}_f(S)$.*

Proof. Let $s_1 = 1 + 2^{k-f+1} - 2^{1-f}$, $s_i = 2^{-F} - 2^{1-f}$ for $1 < i < n$, and $s_n = 2^{-F-1} - 2^{-f} - 2^{-f-1}$. Then for $1 \leq i < n$, $t_i = 1 + 2^{k-f+1} - i2^{1-f}$ (easy induction).

[1] This implies $f \geq 2$ since for $f = 1$ all non-zero numbers have an odd mantissa.

Since $n-1 = 2^k$, we get $t_{n-1} = 1 + 2^{k-f+1} - 2^k 2^{1-f} = 1$, and each of the $n-2$ summations gives an error of $\frac{1}{2}\mathrm{ulp}_F(t_1)$. The total error on S is therefore $\frac{1}{2}(2^k-1)\mathrm{ulp}_F(t_1) + \frac{1}{2}\mathrm{ulp}_F(t_n) + \frac{1}{2}\mathrm{ulp}_f(S) = (2^k-1)\mathrm{ulp}_F(t_n) + \frac{1}{2}\mathrm{ulp}_F(t_n) + \frac{1}{2}\mathrm{ulp}_f(S) = (\frac{3}{2} - \frac{1}{2^{k+1}})\mathrm{ulp}_f(S)$. $\square$

Here is an example for $f = 7$, $F = 10$, $n = 1 + 2^3 = 9$:

$$
\begin{array}{rlrl}
s_1 = t_1 = & 1.000111000 & s_6 = & -\ 0.0000001111000 \\
s_2 = & -\ 0.0000001111000 & t_6 = & 1.000010000 \\
t_2 = & 1.000110000 & s_7 = & -\ 0.0000001111000 \\
s_3 = & -\ 0.0000001111000 & t_7 = & 1.000001000 \\
t_3 = & 1.000101000 & s_8 = & -\ 0.0000001111000 \\
s_4 = & -\ 0.0000001111000 & t_8 = & 1.000000000 \\
t_4 = & 1.000100000 & s_9 = & -\ 0.0000001011100 \\
s_5 = & -\ 0.0000001111000 & t_9 = & 0.1111110100 \\
t_5 = & 1.000011000 & S = & 0.1111110 \\
 & & \text{Correct Sum} = & 0.111111101110
\end{array}
$$

The example from Lemma 6 corresponds to the first case of the proof of Theorem 2, i.e. $E(t_j) \le E(t_i)$ for $j \ge i$. The following example is a worst-case for the second case of that proof, where the exponent increases at some point after index i.

Consider n even, $n \ge 4$, and $f \ge 5$. Start from $s_1 = t_1 = 1 - 2^{-k-1}$, with $s_2 = 2^{-k-2} + 2^{-F-1}$, so we get $t_2 = 1 - 2^{-k-2}$; $s_3 = 2^{-k-2} + 2^{-F}$, which yields $t_3 = 1$; and then alternatively add $2^{-k-2} + 2^{-k-3} + 2^{-F}$ and subtract $2^{-k-2} + 2^{-k-3} - 2^{-F}$. We get $t_4 = t_6 = ... = 1 + 2^{-k-2} + 2^{-k-3}$, while $t_3 = t_5 = t_7 = ... = t_{n-1} = 1$. Finally take $s_n = -2^{-k-2} + 2^{-F-1}$, which gives $t_n = 1 - 2^{-k-2}$. The total error is $(n-2)2^{-F} = (n-2)\mathrm{ulp}_F(t_n)$, which gives a final error of $(\frac{3}{2} - \frac{1}{2^k})\mathrm{ulp}_f(S)$ for $n = 2^k + 1$.

5 Proof of Lemma 5

The lemma that we actually proved in Coq is written somewhat differently. Since Coq has no knowledge of the definition of i, i', t_j and so forth[2] we had to abstract the property. We define for that purpose the `add` subroutine. Let Acc be a F-bit floating-point number and l a list of n f-bit floating-point

[2] and trying to forcibly define them in Coq is difficult.

numbers with non-increasing exponents. We define $add(Acc, l)$:

Subroutine add(Acc, l)
$t_0 \leftarrow Acc$
for $i \leftarrow 1$ **to** n
  $t_i \leftarrow \circ_F(t_{i-1} + l_i)$
Return t_n.

Thus summing l with Algorithm DemmelHida is the same as $\circ_f(add(0, l))$. We define as well $radd(Acc, l)$ [3] as being the sum of Acc and the numbers in l with infinite precision.

Let p, q be lists, we also define $p \mid\mid q$ as the concatenation of p and q and $|p|$ as the number of elements in p. We note $[]$ the empty list and $[a]$ the list consisting of the floating-point number a.

Denoting by Acc the value of $t_{i'}$, and by $l := [s_{i'+1}, ..., s_n]$ the list of remaining summands, Lemma 5 now reads:
Lemma 5 *Let l a non empty list of f-bit floating-point numbers with non increasing exponents, and Acc an F-bit floating-point number. Assume*

$$\forall(p, q),\ l = (p \mid\mid q) \textit{ and } p \neq [] \Rightarrow \mathrm{ulp}_F(add(Acc, p)) < \mathrm{ulp}_F(Acc) \tag{3}$$

$$\forall s \in l, \mathrm{ulp}_f(s) < \mathrm{ulp}_F(Acc). \tag{4}$$

and

$$|l| \leq 2^k$$

Then we have :

$$2^f|add(Acc, l) - radd(Acc, l)| \leq |add(Acc, l)| - |Acc| + |l|2^{f-1}\mathrm{ulp}_F(Acc).$$

The proof is done by induction on $|l|$. First assume $|l| = 1$, and note $l = [s_1]$. The lemma now reads:

$$2^f| \circ_F (Acc + s_1) - (Acc + s_1)| \leq | \circ_F (Acc + s_1)| - |Acc| + 2^{f-1}\mathrm{ulp}_F(Acc).$$

We split our proof in two cases: either $\mathrm{ulp}(s_1) = \frac{1}{2}\mathrm{ulp}_F(Acc)$, or $\mathrm{ulp}(s_1) < \frac{1}{2}\mathrm{ulp}_F(Acc)$ (remember $\mathrm{ulp}(s_1) < \mathrm{ulp}_F(Acc)$ since $Acc = t_{i'}$, and $\mathrm{ulp}_F(t_{i'}) =$

[3] "real add"

$\text{ulp}_F(t_i))$. If $\text{ulp}(s_1) = \frac{1}{2}\text{ulp}_F(Acc)$, then $\text{ulp}(s_1) \geq \text{ulp}_F(\circ_F(Acc + s_1))$, and the sum is exact by Lemma 1. We get:

$$|Acc| - 2^{f-1}\text{ulp}_F(Acc) \leq |Acc + s_1|.$$

Since $|s_1| \leq 2^{f-1}ulp_F(Acc)$, the result follows.

Now assume $\text{ulp}(s_1) < \frac{1}{2}\text{ulp}_F(Acc)$. We have to prove:

$$2^{f-1}[\text{ulp}_F(\circ_F(Acc + s_1)) - \text{ulp}_F(Acc)] + |Acc| \leq |\circ_F (Acc + s_1)|.$$

From the hypothesis we have — since ulps are powers of two — $\text{ulp}_F(\circ_F(Acc+ s_1)) \leq \frac{1}{2}\text{ulp}_F(Acc)$, so the goal is rewritten as

$$|Acc| - 2^{f-2}\text{ulp}_F(Acc) \leq |\circ_F (Acc + s_1)|,$$

which follows from Lemma 3, since $|s_1| < 2^{f-2}\text{ulp}_F(Acc)$ (Acc and s_1 are necessary of different signs since their rounded sum has a smaller exponent than Acc).

Consider now $l = p \mid\mid [a]$ where a is the last added floating-point number. Let $t = add(Acc, p)$. We split our proof again in two subcases and assume first that the last summation is exact, i.e. $t + a = \circ_F(t + a)$. The goal reads:

$$2^f|t - radd(Acc, p)| \leq |t + a| - |Acc| + (|p| + 1)2^{f-1}\text{ulp}_F(Acc).$$

Since $|t| \leq |t+a|+|a|$, we get using the induction hypothesis $2^f|t-radd(Acc, p)| \leq |t| - |Acc| + |p|2^{f-1}\text{ulp}_F(Acc)$:

$$2^f|t - radd(Acc, p)| \leq |t + a| + |a| - |Acc| + |p|2^{f-1}\text{ulp}_F(Acc),$$

and Hypothesis (4) concludes this subcase.

Assume now that the last computation is inexact. Then $\text{ulp}_f(a) < \text{ulp}_F(\circ_F(t+ a)) < \frac{1}{2}\text{ulp}_F(Acc)$ by Cor. 1 and Hypothesis (3).

$$|\circ_F (t + a) - radd(Acc, p \mid\mid [a])| \leq |\circ_F (t + a) - (t + a)| + |t - radd(Acc, p)|.$$

Using the induction hypothesis and the correct rounding of the addition, we can write :

$$\begin{aligned}2^f|\circ_F (t + a) - radd(Acc, p \mid\mid [a])| \leq\, & 2^{f-1}\text{ulp}_F(\circ_F(t + a)) - |Acc| \\ & +|t| + |p|2^{f-1}\text{ulp}_F(Acc),\end{aligned}$$

Using successively Lemma 1 and Hypothesis (3) we have

$$\mathrm{ulp}_f(a) \leq 2^{-1}\mathrm{ulp}_F(\circ_F(t+a)) \leq 2^{-2}\mathrm{ulp}_F(Acc)$$

$$|a| < 2^{f-2}\mathrm{ulp}_F(Acc) = 2^{E(Acc)-k-2}.$$

From Lemma 4 we know that the hypothesis of Lemma 3 with $2^l = 2^{f-2}\mathrm{ulp}_F(Acc)$ holds since $E(t) \geq E(Acc) - k > E(Acc) - k - 2$, and we get:

$$|t| \leq |\circ_F(t+a)| + 2^{f-2}\mathrm{ulp}_F(Acc)$$

and the goal is now trivial with the help of Hypothesis (3). $\square$

The proof of Lemma 5 was written in the Coq language, using version 7.4 of the `Float` library developed by Laurent Théry and Laurence Rideau, and maintained by Sylvie Boldo [1]. It consists in approximatively 900 lines in the Coq language (including the statement of the different hypotheses and other lemmas), and is available at `http://www.loria.fr/~fousse/rnc5.v`. The definition of the lemma itself is written in 13 lines, its proof takes 430 lines. We used a special trick in Coq that permitted us to assume some properties and leave them to be proved later. More time could therefore be spent on the main proof, and since the properties that are currently left assumed are obvious for the human reader this does not hinder the correctness of the result. Such properties for example read:

- an ulp is always positive,
- $2^f - 1 \geq 2^{f-1}$,
- $2^f - 1 \geq 0$.

The holes however need te be eventually filled so that the human proofreading is confined to the understanding of the definition of the final theorem in Coq.

6 Final Remarks

The use of a formal proof assistant was very profitable to the human-readable form of the proof: it helped us focus on the main result and define only the lemmas that are actually used in the proof. Coq is rather demanding in the way the objects are defined and their properties, even simple ones, can be asserted. This is a good thing since the lemmas are naturally abstracted and no hypothesis can be neglected in the formulation of the lemma. The reusability and the confidence one can have in the correctness of the results proved is then increased [4]. For example, following a suggestion of Hida [4], it would be interesting to extend the error bound and its proof to an arbitrary radix, instead of just considering binary floating-point numbers. If the disturbing lack of clue of Coq may be seen as the fact that Coq-proven properties are indeed well proven, this is however a burden when dealing with floating-point related properties which contain lots of inequalities. We strongly encourage Coq developers to add better computing abilities to their theorem prover because it would spread its use in computer arithmetic where certified algorithms are badly needed.

References

[1] Daumas, M., Rideau, L., and Théry, L. A Generic Library for Floating-Point Numbers and Its Application to Exact Computing. In *Theorem Proving in Higher Order Logics: 14th International Conference* (Sept. 2001), no. 2152 in LNCS, Springer-Verlag.

[2] Demmel, J., and Hida, Y. Fast and accurate floating point summation with application to computational geometry. *Numerical Algorithms*. To appear. Presented at 10th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN 2002).

[3] Demmel, J., and Hida, Y. Accurate floating point summation. `http://www.cs.berkeley.edu/~demmel/AccurateSummation.ps`, May 2002. 37 pages.

[4] Hida, Y. Personal communication, Jan. 2004.

[5] IEEE standard for binary floating-point arithmetic. Tech. Rep. ANSI-IEEE Standard 754-1985, New York, 1985. approved March 21, 1985: IEEE Standards Board, approved July 26, 1985: American National Standards Institute, 18 pages.

[6] Kulisch, U., and Bohlender, G. Formalization and implementation of floating-point matrix operations. *Computing 16* (1976), 239–261.

[7] Priest, D. M. Algorithms for arbitrary precision floating point arithmetic. In *Proc. of the 10th Symposium on Computer Arithmetic* (Grenoble, France, 1991), P. Kornerup and D. Matula, Eds., IEEE Computer Society Press, pp. 132–144.

[8] Russinoff, D. M. A mechanically checked proof of correctness of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7 processor. *Journal of Computation and Mathematics 1* (Dec. 1998). `www.onr.com/user/russ/david/k7-div-sqrt.html`.

[9] The Coq Development Team, LogiCal Project. *The Coq Proof Assistant: Reference Manual*, version 7.4 ed., 2003. `http://coq.inria.fr/doc/main.html`.

[4] since the confidence is generally directly proportional to the amount of work one needs to put in the proof.