How to Compute the Area of a Triangle: a Formal Revisit with a Tighter Error Bound

Sylvie Boldo

How to Compute the Area of a Triangle: a Formal Revisit with a tighter error bound

Sylvie Boldo, *Member, IEEE*

Abstract—Mathematical values are usually computed using well-known mathematical formulas without thinking about their accuracy, which may turn awful with particular instances. This is the case for the computation of the area of a triangle. When the triangle is needle-like, the common formula has a very poor accuracy. Kahan proposed in 1986 an algorithm he claimed correct within a few ulps. Goldberg took over this algorithm in 1991 and gave a precise error bound. This article presents a formal proof of this algorithm, investigations in case of underflow and a new improvement of its error bound.

Index Terms—floating-point arithmetic, formal proof, Coq, triangle, underflow

1 Introduction

Floating-point (FP) arithmetic is seen as intricate because too few people have sufficient knowledge to understand how it works. For people having been only trained with mathematics, facts such that $(x+y)+z$ may be different from $x+(y+z)$ for certain values or the fact that there exists x such that $x \neq 0$, but $x^2 = 0$ is beyond comprehension. This is the reason why mathematical formulas are most of the time programmed as they stand in mathematical textbooks.

We are interested here in computing the area of a triangle, given its side lengths as FP numbers. This is especially difficult (using FP computations) for needle-like triangles like the one in Figure 1.

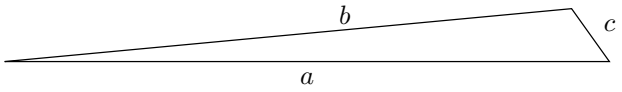

Fig. 1. A Needle-Like Triangle

The common formula to compute the area is two millennia old and is attributed to Heron of Alexandria:

$$\Delta = \sqrt{s\ (s-a)\ (s-b)\ (s-c)}$$

$$\text{where } s = \frac{a+b+c}{2}$$

This formula is known to be inaccurate using floating-point arithmetic since the 80s. This has been first studied by Kahan [1]. He gave examples of incorrect computations: either the result was very wrong or the computation was stopped due to a negative square root, created by round-off errors. Kahan also proposed an algorithm that behaves correctly using floating-point arithmetic, that is to use the following formula:

$$\frac{1}{4}\sqrt{(a+(b+c))\ (c-(a-b))\ (c+(a-b))\ (a+(b-c))}$$

The parenthesis are not to be removed to guarantee that the square root will carry out on a non negative number. Kahan claimed the safety and the accuracy of the result within a few ulps by giving concise and precise arguments.

A little later, Goldberg presented this algorithm in his famous article "What Every Computer Scientist Should Know About Floating-Point Arithmetic" [2]. More than the algorithm, he gave a precise error bound, but gave no proof or hint of the reason why it behaves so well. He claimed that, given a machine epsilon $\varepsilon \leq .005$, the rounding error was at most $11\,\varepsilon$. Recently, this error bound was improved by the author to 6.625ε [3].

To get a high guarantee on mathematical results or programs, formal methods have recently developed. This of course includes floating-point arithmetic that has been formalized since 1989 in order to formally prove hardware components or algorithms [4], [5], [6]. This algorithm is a good test case for formal proof checking. The first reason is that this is common knowledge, so it is believed both that it cannot be false and that it can hardly be enhanced. A recent example of the computation of the discriminant [7] has shown that pen-and-paper proofs may miss difficulties: here the fact that the floating-point test may be mistaken [8]. The second reason is that this is an uncommon kind of proof compared to what has been proved before. It is somewhat easier: this is forward error analysis with few floating-point cunning facts. Formal proofs are usually applied to more complex and trickier algorithms. Anyway, even if this algorithm and these proofs are decade old, we were able to notably improve the error bound.

From the algorithm point of view, this work gives a high guarantee of its correctness and gives precise

• *S. Boldo is with Inria Saclay–Île-de-France, LRI, CNRS UMR 8623, Université Paris-Sud, Bât 650 (PCRI), Orsay, F-91405, France.*
E-mail: sylvie.boldo@inria.fr

This work was supported by the VERASCO project (ANR-11-INSE-003) of the French National Agency for Research (ANR).

hypotheses on the radix and the needed precision. We are also sure to take into account second-order terms for the round-off error, which are usually dismissed. Here, they cannot be dismissed and must therefore be bounded, even coarsely. From the formal methods point of view, we will base our proof on the recent Flocq library [9] that has not yet been thoroughly used for floating-point algorithms. Flocq is a formalization in Coq that offers a multi-radix and multi-precision formalization for various floating- and fixed-point formats (including FP with or without gradual underflow) with a comprehensive library of theorems. Its usability and practicality have to be established against test-cases.

Another point is the difficulty in handling subnormals. Most pen-and-paper proofs assume there is neither underflow, nor overflow. This is a very strong hypothesis that greatly simplifies the proofs. Unfortunately, it may be difficult to give conditions beforehand or to check afterwards that this hypothesis is fulfilled. This is the reason why we take care of subnormals and their consequences for round-off errors.

This article is organized as follows. Section 2 presents the proof when no underflow occur, the improvement of the error bound and the formal demonstration. Section 3 presents the modification when taking gradual underflow into account. Section 4 presents the proved C program with annotations stating its precise specifications, including overflow considerations.

Notations:

The side lengths of the triangle will be denoted by a, b and c and are assumed to be exact FP numbers. We have ordered these lengths beforehand, so that $0 \leq c \leq b \leq a$. As these values represent a (possibly degenerate) triangle, basic geometry requires that $a \leq b + c$.

We will denote the radix by β, an integer greater than 1. The precision of the floating-point format will be denoted by p and will be greater than 1. The minimal exponent will be denoted by E_i: this means that the smallest positive FP number (the smallest subnormal) is β^{E_i}. We will denote by $\circ$ the default rounding mode (rounding to nearest, ties to even), and we will denote by $\oplus$ the FP addition, $\ominus$ the FP subtraction, and $\otimes$ the FP multiplication. We will denote by ε the machine epsilon, that is to say the relative error bound for normal numbers in rounding to nearest, which is $\varepsilon = \frac{\beta^{1-p}}{2}$.

We will denote by $\mathfrak{C}(x)$ (for correct) the exact value of a floating-point x, meaning the value it would have had without any rounding. In particular, $\mathfrak{C}(a) = a$, and $\mathfrak{C}(a \oplus b) = a + b$.

All the theorems stated in this article correspond to one or several Coq theorems. This development is available on http://www.lri.fr/~sboldo/research.html.

Algorithm

For the sake of readability, here is Kahan's algorithm with temporary variables:

$$\begin{aligned} t_1 &= a \oplus (b \oplus c) \\ t_2 &= a \oplus (b \ominus c) \\ t_3 &= c \oplus (a \ominus b) \\ t_4 &= c \ominus (a \ominus b) \\ M &= ((t_1 \otimes t_2) \otimes t_3) \otimes t_4 \\ \Delta &= \circ\left(\circ\left(\frac{1}{4}\right) \circ (\sqrt{M})\right) \end{aligned}$$

Note that Kahan's algorithm computes $t_1 \otimes t_2 \otimes t_3 \otimes t_4$ without parenthesis, but we chose to parenthesize the multiplication for reasons explained later in Section 3.4.

2 When no Underflow Occurs

2.1 Hypotheses

We first consider a floating-point format on p bits with unbounded exponent range. It exactly corresponds to the very common "provided no underflow, or overflow occur". This greatly simplifies the proof for a beginning, and helps to get a tight error bound without having more to take subnormal into account. This corresponds to the FLX format defined in Flocq. A value is said to fit in the format when it can be represented by a floating-point number, and is therefore computed without rounding.

Even if we consider a generic radix, we have in thought that β will be 2 or 10 (or even 4 or 16), therefore we have assumed that $\frac{1}{4}$ fits in the format. We also proved this was correct for $\beta = 2$ and $\beta = 10$ as soon as $p \geq 2$.

We also assume that the precision is not too small. More precisely, we require that the machine epsilon, that is to say $\varepsilon = \frac{\beta^{1-p}}{2}$ is smaller or equal to $\frac{1}{100}$. This is guaranteed for a precision greater than 6 with $\beta = 2$ and greater than 2 with $\beta = 10$. This is summarized as follows:

Set of Hypotheses 1 *We assume in this section:*

- *the exponent range is unbounded,*
- $\frac{1}{4}$ *fits in the format,*
- $\varepsilon \leq \frac{1}{100}$,
- $0 \leq c \leq b \leq a \leq b + c$.

2.2 Non-negativity

First, we will prove that the computation will not fail due to taking the square root of a negative number. This requires to prove that $M \geq 0$. As roundings are monotone, it is sufficient to prove that all t_is are non-negative. The monotonicity of the rounding will be used thoroughly and is sufficient to prove all the required inequalities, given the assumptions on a, b, and c. For example, to prove that $t_4 \geq 0$, it is sufficient to prove that $c - (a \ominus b) \geq 0$, which is equivalent to $c \geq a \ominus b$. By the monotonicity of the rounding and as c fits in the format, it is sufficient that $c \geq a - b$ which is exactly the assumption $a \leq b + c$.

2.3 Exact computations

One of the key point in the error bound proof is the fact that $a \ominus b = a - b$. This subtraction could have created a large relative error, but is in fact exact due to the assumptions on the inputs. More precisely, we use Sterbenz theorem on exact subtraction [10]. There is left to prove that $\frac{b}{2} \leq a \leq 2 \cdot b$. We know that $b \leq a$ and that $a \leq b + c \leq 2 \cdot b$ as $b \leq c$. So $a \ominus b$ is computed without error.

A new improvement of the error bound is due to the fact that t_4 is also computed exactly. Indeed, $t_4 = c \ominus (a \ominus b)$ therefore $t_4 = \circ(c - (a - b))$ from the previous remark. As a fits in the format, it can be represented as $n_a \beta^{m_a}$ with $|n_a| < \beta^p$ and in the same way for b and c. As $0 \leq c \leq b \leq a$, if we choose the smallest possible exponents, we know that $m_c \leq m_b \leq m_a$. So the value $c - (a - c)$ can be represented as $n \times \beta^{m_c}$. Moreover,

$$\begin{aligned} |n| &= |c - (a - b)|\beta^{-m_c} \\ &= (c - (a - b))\beta^{-m_c} \\ &< c\beta^{-m_c} = n_c < \beta^p \end{aligned}$$

Therefore $c - (a - b)$ fits in the format.

Theorem 1 (t4_exact_flx) *We assume the set of hypotheses 1. Then,* $t_4 = c - (a - b)$.

2.4 Error Lemmas

All the remaining operations may not be exact and we have to bound their relative errors to get a final relative error. We will denote by $\mathrm{err}(x, y, e)$ the mathematical inequality

$$|x - y| \leq e \cdot |y|.$$

It will mean that x is an approximation of the exact value y with a relative error e.

The first two theorems may seem silly. Their use will be explained in Section 2.5.

Theorem 2 (err_aux) *Given* x, y, e_1, e_2, *if* $e_1 \leq e_2$ *and* $\mathrm{err}(x, y, e1)$, *then* $\mathrm{err}(x, y, e2)$.

Theorem 3 (err_0) *Given* x, *we have* $\mathrm{err}(x, x, 0)$.

We will now express floating-point operations with this error. For now, this is naive forward error analysis.

Theorem 4 (err_init_flx) *We assume the set of hypotheses 1. Given* x, *we have* $\mathrm{err}(\circ(x), x, \varepsilon)$.

Remember we assume an unbounded exponent range. For all real numbers, the error can be seen as a relative error bound bounded by ε. Note this was an application of a standard theorem from Flocq.

Theorem 5 (err_add_flx) *We assume the set of hypotheses 1. Given* $x_1, y_1, e_1, x_2, y_2, e_2$, *if* $\mathrm{err}(x_1, y_1, e_1)$, *and* $\mathrm{err}(x_2, y_2, e_2)$, *and* $0 \leq y_1$ *and* $0 \leq y_2$, *then*

$$\mathrm{err}(x_1 \oplus x_2, y_1 + y_2, \varepsilon + (1 + \varepsilon) \cdot \max(e_1, e_2)).$$

There is nothing new here: this is a typical theorem from forward analysis. Given the fact that y_1 and y_2 are non-negative, the error of an addition is ε plus the maximum of the errors of the inputs multiplied by $1+\varepsilon$. The proof is straightforward.

Theorem 6 (err_mult_flx) *We assume the set of hypotheses 1. Given* $x_1, y_1, e_1, x_2, y_2, e_2$, *if* $\mathrm{err}(x_1, y_1, e_1)$, *and* $\mathrm{err}(x_2, y_2, e_2)$ *then*

$$\mathrm{err}(x_1 \otimes x_2, y_1 \cdot y_2, \varepsilon + (1 + \varepsilon) \cdot (e_1 + e_2 + e_1 \cdot e_2)).$$

There is nothing new here either. We take care of the second order term $e_1 \cdot e_2$ even if it will probably be negligible. The proof is also straightforward.

Theorem 7 (err_sqrt_flx) *We assume the set of hypotheses 1. Given* x, y, e, *if* $0 \leq y$, *and* $e \leq 0.5$ *and* $\mathrm{err}(x, y, e)$, *then*

$$\mathrm{err}\left(\circ\left(\sqrt{x}\right), \sqrt{y}, \varepsilon + (1 + \varepsilon) \cdot \left(\frac{e}{2} + \frac{e^2}{4}\right)\right).$$

This is the theorem that allows us to improve over Goldberg's bound. Probably, his point of view was to consider that $|\sqrt{1+h} - 1| \leq |h|$ for $|h| \leq 1$ and this gives an error bound which is $\varepsilon + (1+\varepsilon) \cdot e$ and gives $11\,\varepsilon$ at the end. But in fact, $|\sqrt{1+h} - 1| \approx \frac{|h|}{2}$ but may be greater for negative h, thus the second-order term. So if we assume h is small, we prove that $|\sqrt{1+h} - 1| \leq \frac{|h|}{2} + \frac{|h|^2}{4}$. The proof was quite tedious (90 lines of Coq) as it is based on manual manipulations of real expressions involving square roots.

2.5 Main Proof

We now have all the necessary theorems to do forward error analysis in Coq on this algorithm. This was expected to be very tedious as formulas get rather complicated, but this was rather easy. The idea is to use the `eapply` Coq tactic that applies partly a theorem: this means that some variables remain un-instantiated

for some time during the proof, and are noted with a `?` followed by a number. For example to prove $x \leq y$, we first prove that $x \leq$ `?1375` and then that `?1375` $\leq y$. Here, we want to bound the error of the computation of M. As we do not know beforehand the error of M and we do not want to compute it exactly by hand, we use the err_aux theorem to get an unknown value as the error bound of M. We then have to prove that $\mathrm{err}(M, \mathfrak{C}(M), \texttt{?1352})$. Then, as M is the result of a multiplication, we apply the err_mult theorem. So the unknown value is partly instantiated: the error bound of M is $\varepsilon + (1+\varepsilon) \cdot (\texttt{?1359} + \texttt{?1360} + \texttt{?1359} \cdot \texttt{?1360}))$ and we have to prove that $\mathrm{err}(t4, c-(a-b), \texttt{?1360})$ and that $\mathrm{err}((t_1 \otimes t_2) \otimes t_3, (\mathfrak{C}(t_1) \cdot \mathfrak{C}(t_2)) \cdot \mathfrak{C}(t_3)), \texttt{?1359})$. Step by step, depending on the last FP operation, we instantiate unknown values, and sometimes create them. We use Theorems err_init and err_0 to solve the simplest goals and we take advantage of the fact that $a \ominus b = a - b$ and that $t_4 = c - (a - b)$. At the end, we get a large formula for the error of M:

$$\begin{gathered}\varepsilon + (1+\varepsilon) \cdot (\varepsilon + (1+\varepsilon) \cdot (\varepsilon + (1+\varepsilon) \cdot \\ (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot \\ (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon)) + \varepsilon + (\varepsilon + (1+\varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + \\ (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon))) \cdot \varepsilon) + 0 + \\ (\varepsilon + (1+\varepsilon) \cdot (\varepsilon + (1+\varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \\ + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon)) + \varepsilon + (\varepsilon + (1+\varepsilon) \cdot \\ (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot \\ (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon))) \cdot \varepsilon)) \cdot 0)\end{gathered}$$

but we did not have to give it directly to Coq, as was expected! The value is indeed needed to be exactly put in the right form for theorems to be applied.

We can then make the prover reorganize the formula and we get:

$$\varepsilon^8 + \frac{15}{2} \cdot \varepsilon^7 + 25 \cdot \varepsilon^6 + \frac{97}{2} \cdot \varepsilon^5 + 60 \cdot \varepsilon^4 + \frac{97}{2} \cdot \varepsilon^3 + 25 \cdot \varepsilon^2 + \frac{15}{2} \cdot \varepsilon$$

We know that ε is small and that the terms with the highest exponents will be negligible. As we required ε to be smaller than 0.01, we were able with some effort to bound this value by $\frac{15}{2}\varepsilon + 26\varepsilon^2$. To prove this, we first detailed all the steps, making this a long (80 lines) tedious proof. Now, it is done in about 15 lines as we rely on the `interval` tactic [11], that is able to prove it if we manually do the first steps of the proof.

Theorem 8 (err_M_flx) *We assume the set of hypotheses 1. We have*

$$\mathrm{err}(M, \mathfrak{C}(M), \frac{15}{2}\varepsilon + 26\varepsilon^2).$$

Note that the order of the t_i does not matter here: we may have chosen $M = ((t_2 \otimes t_4) \otimes t_1) \otimes t_3$, it would have given the same error bound.

2.6 Correctness Theorem with unbounded exponent range

To end the proof, we just have to go on with the forward analysis with the square root computation and the multiplication by $\frac{1}{4}$. This last value is computed correctly as it is assumed to fit in the format. Thus the bound on the round-off error of Δ:

Theorem 9 (err_Δ_flx) *We assume the set of hypotheses 1. We have*

$$\mathrm{err}(\Delta, \mathfrak{C}(\Delta), \frac{23}{4}\varepsilon + 38\varepsilon^2).$$

Instead of the $11\,\varepsilon$, we are able to formally guarantee that the relative error is $5.75\,\varepsilon$ (plus the second-order terms).

We can still tighten this bound in radix 2: in this case, multiplying by $\frac{1}{4}$ is exact, therefore the last computation does not create any round-off error, and we can prove:

Theorem 10 (err_Δ_flx_radix2) *We assume the set of hypotheses 1 and $\beta = 2$. We have*

$$\mathrm{err}(\Delta, \mathfrak{C}(\Delta), \frac{19}{4}\varepsilon + 33\varepsilon^2).$$

Instead of the $11\,\varepsilon$, we formally guarantee that the relative error is $4.75\,\varepsilon$ (plus the second-order terms). This means a nearly 60 % better bound on the relative round-off error from Goldberg's bound.

3 Taking Gradual Underflow into Account

Unfortunately, the exponent range is limited: the IEEE-754 standard [12] precisely states what are the minimal and maximal exponents allowed in the `binary32` and `binary64` formats. In this Section, we will only consider gradual underflow. Overflows will be looked into in the next Section. We will have a minimal exponent and take into account the fact that subnormal results may appear and produce a huge relative error bound. This corresponds to the FLT format of Flocq.

3.1 Hypotheses

Set of Hypotheses 2 *We assume in this section:*

- *gradual underflow with E_i as minimal exponent,*
- $E_i \leq -3 - p$,
- *no upper bound on the exponent,*
- $\frac{1}{4}$ *fits in the format,*
- $\varepsilon \leq \frac{1}{100}$,
- $0 \leq c \leq b \leq a \leq b + c$.

The last four hypotheses were assumed in the previous Section. The second hypothesis is true in all reasonable formats. It only states that subnormal numbers are small and is equivalent to have the smallest positive normal number smaller or equal to β^{-4}.

3.2 Exact computations

First and for the same reason as before, we have proved that $M \geq 0$ and that $a \ominus b = a - b$. These proofs are based on the monotonicity of the rounding and on Sterbenz theorem, which are still valid with gradual underflow.

Now, we want also to prove that t_4 is also computed exactly. As before, $t_4 = c \ominus (a \ominus b) = \circ(c - (a - b))$. The proof follows the same pattern, but takes into account the minimal exponent. As a fits in the format, it can be represented as $n_a \beta^{m_a}$ with $|n_a| < \beta^p$ and $m_a \geq E_i$, and in the same way for b and c. As $0 \leq c \leq b \leq a$, if we choose the smallest possible exponents (greater or equal to E_i), we know that $m_c \leq m_b \leq m_a$. So the value $c-(a-c)$ can be represented as $n \times \beta^{m_c}$. As before, we prove that $|n| < \beta^p$. So we both have that $m_c \geq E_i$ and that $|n| < \beta^p$ therefore $c - (a - b)$ fits in the format.

Theorem 11 (t4_exact_flt) *We assume the set of hypotheses 2. Then,* $t_4 = c - (a - b)$.

3.3 Error Lemmas

We are of course trying to use as much as possible the previous formal proofs, but some theorems are not valid any more (for example err_init). Here are the changes when subnormal may appear.

For the addition, it is well-known that, if the result is subnormal, it is exact. Therefore the ε bound also holds here. Thus we exactly have the same formula as in Theorem err_add_flx:

Theorem 12 (err_add_flt) *We assume the set of hypotheses 2. Given* $x_1, y_1, e_1, x_2, y_2, e_2$, *if* $\mathrm{err}(x_1, y_1, e_1)$, *and* $\mathrm{err}(x_2, y_2, e_2)$, *and* $0 \leq y_1$ *and* $0 \leq y_2$, *then*

$$\mathrm{err}(x_1 \oplus x_2, y_1 + y_2, \varepsilon + (1 + \varepsilon) \cdot \max(e_1, e_2)).$$

Concerning the multiplication, we cannot dodge the subnormals as with the addition. In the case where the output of the FP multiplication is subnormal, the relative error bound gets huge. To prevent this, we will have to prove that gradual underflow will not happen to guarantee the wanted error bound. We recall that β^{E_i} is the smallest positive subnormal number.

Theorem 13 (err_mult_flt) *We assume the set of hypotheses 2. Given* $x_1, y_1, e_1, x_2, y_2, e_2$, *if* x_1 *and* x_2 *fit in the format, if* $\mathrm{err}(x_1, y_1, e_1)$, *and* $\mathrm{err}(x_2, y_2, e_2)$, *and if* $\beta^{E_i+p-1} < |x_1 \otimes x_2|$, *then*

$$\mathrm{err}(x_1 \otimes x_2, y_1 \cdot y_2, \varepsilon + (1 + \varepsilon) \cdot (e_1 + e_2 + e_1 \cdot e_2)).$$

As for the square root, there is nearly no subnormal problem. If x is in the format, its square root cannot be subnormal, except for zero. When $x = 0$, the next theorem is not valid, therefore we put an hypothesis to prevent that, knowing it will be easy to prove in our context. We also could have proved that $x \neq 0$ was sufficient.

Theorem 14 (err_sqrt_flt) *We assume the set of hypotheses 2. Given* x, y, e, *if* $0 \leq y$, *and* $e \leq 0.5$, *and* $\beta^{E_i+p-1} < \circ(\sqrt{x})$, *and* $\mathrm{err}(x, y, e)$, *then*

$$\mathrm{err}\left(\circ\left(\sqrt{x}\right), \sqrt{y}, \varepsilon + (1 + \varepsilon) \cdot \left(\frac{e}{2} + \frac{e^2}{4}\right)\right).$$

We have all the basic blocks to do forward analysis on a FP format with gradual underflow. But, as far as the multiplication are concerned, we have to prove that no subnormal will appear.

3.4 Ordering the t_is

As explained, we will need to prove that no subnormal is created. This means we need either to know beforehand or to detect afterwards if subnormals appear. In this case, the detection afterwards was a better solution. The reason is that the hypotheses needed on a, b and c to guarantee this fact would have been very strong, and much stronger than the chosen hypothesis on the result. More precisely, we give a limit value for the result of the algorithm: if it is above this limit, we are sure that no subnormal was created and that the error bound holds. If it is under the limit, a subnormal may have appeared and the relative error bound may be much greater than the expected bound.

We then need to detect any underflow. As the multiplications are the only problems, we have to look into the computation of M and we may try to organize the t_i as we want. As explained before, the original algorithm by Kahan did not put any parenthesis and was stated as $t_1 \otimes t_4 \otimes t_3 \otimes t_2$ while we choose $M = ((t_1 \otimes t_2) \otimes t_3) \otimes t_4$. The reason is that we ordered the t_i by magnitude. More precisely, we proved that:

Theorem 15 *We assume the set of hypotheses 2. We have* $0 \leq t_4 \leq t_3 \leq t_2 \leq t_1$.

This is useful as it means that a subnormal result will not disappear. More precisely, if the result of a multiplication is a subnormal, the results of the following FP multiplications will also be subnormal. This means that, if a subnormal appear, then M will be a subnormal, which is very easy to check. The idea is to forbid the cases where an internal multiplication creates a subnormal, but this fact cannot be detected at the end of the computation.

For that, we will use the following theorem:

Theorem 16 (subnormal_aux) *We assume the set of hypotheses 2. Given x and y, we assume that x fits in the format and that $\beta^{E_i+p-1} < |x \otimes y|$. We also assume that, if $|x| \leq 1$, then $|y| \leq 1$. Then $\beta^{E_i+p-1} < |x|$.*

The idea of the proof is quite simple. By the absurd, we assume that $|x| \leq \beta^{E_i+p-1}$. From the hypothesis on E_i, it means that $|x| \leq 1$ and so that $|y| \leq 1$. Then we have $|x \cdot y| \leq \beta^{E_i+p-1} \cdot 1 = \beta^{E_i+p-1}$. As β^{E_i+p-1} fits in the format and by monotonicity of the rounding, we have that $|x \otimes y| \leq \beta^{E_i+p-1}$ which is absurd.

This theorem allows us to prove that, if M is normal, then $(t_1 \otimes t_2) \otimes t_3$ is also normal, which also implies that $t_1 \otimes t_2$ is normal. Then all the multiplication results are normal and the err_mult_flt Theorem can be applied. In this proof, there are several other goals, namely the ones corresponding to "if $|x| \leq 1$, then $|y| \leq 1$". They are indeed straightforward as the t_i are ordered.

3.5 Correctness Theorem with Gradual Underflow

We can now apply the same kind of proof as in Section 2.5 with forward error analysis, helped by the proof assistant. We of course get the same error bound, provided M is not a subnormal:

Theorem 17 (err_M_flt) *We assume the set of hypotheses 2 and that $\beta^{E_i+p-1} < M$. We have*

$$\mathrm{err}(M, \mathfrak{C}(M), \frac{15}{2}\varepsilon + 26\varepsilon^2).$$

To end the proof, we just have to go on with the forward analysis with the square root computation and the multiplication by $\frac{1}{4}$. This last value is computed correctly as it is assumed to fit in the format. Thus the bound on the round-off error of Δ, provided Δ is big enough (so that M is big enough):

Theorem 18 (err_Δ_flt) *We assume the set of hypotheses 2 and that $\frac{1}{4}\beta^{\lceil \frac{E_i+p-1}{2} \rceil} < \Delta$. We have*

$$\mathrm{err}(\Delta, \mathfrak{C}(\Delta), \frac{23}{4}\varepsilon + 38\varepsilon^2).$$

Instead of the $11\,\varepsilon$ of Goldberg, we are able to formally guarantee that the relative error is 5.75ε (plus the second-order terms) while taking into account gradual underflow.

We can still tighten this bound in radix 2: in this case, multiplying by $\frac{1}{4}$ is exact, therefore the last computation does not create any round-off error, as we are far from the underflow threshold:

Theorem 19 (err_Δ_flt_radix2) *We assume the set of hypotheses 2, that $\beta = 2$, and that $2^{\lceil \frac{E_i+p-1}{2} \rceil - 2} < \Delta$. We have*

$$\mathrm{err}(\Delta, \mathfrak{C}(\Delta), \frac{19}{4}\varepsilon + 33\varepsilon^2).$$

Instead of the $11\,\varepsilon$ of Goldberg, we formally guarantee that the relative error is $4.75\,\varepsilon$ (plus the second-order terms) while taking into account gradual underflow.

4 Program Proof

The preceding proof has several advantages: it is generic in terms of radix and of precision. It has also drawbacks: it does not take overflows into account and it is a Coq proof. As a Coq theorem, it is always difficult to convince people that the program they use fits the Coq theorems. Another difficulty is the hypotheses on the formats (precision, minimal exponent) that may be hidden in the Coq proofs. To check this proof against a real program, we have annotated and proved a real C program. It is a very simple program that only computes `return (0x1p-2*sqrt((a+(b+c))*(a+(b-c))*(c+(a-b))*(c-(a-b))))`.

We use the Frama-C platform[1] to perform formal verification of C programs at the source-code level. Frama-C is an extensible framework that combines static analyzers for C programs, written as plug-ins, within a single tool. In this work, we use the Jessie plug-in for deductive verification. C programs are annotated with behavioral contracts written using the ANSI C Specification Language [13] that tries to be as near C statements as possible. The Jessie plug-in translates them to the Why3 verification platform [14]. Finally, the Why3 platform computes verification conditions from these programs, using traditional techniques of weakest preconditions, and emits them to a wide set of existing theorem provers, ranging from interactive proof assistants to automated theorem provers. In this work, we use the Coq proof assistant, and the automated theorem prover Gappa that uses interval arithmetic to prove properties that occur when verifying numerical applications [15].

The full annotated program is in Figure 2. Here are some details about the annotations. We only consider the `double` type meaning the `binary64` type of the IEEE-754. First, the square root is defined as an external

1. http://www.frama-c.cea.fr/

```
/*@ requires 0 <= x;
  @ ensures \result==\round_double(\NearestEven,\sqrt(x));
  @*/
double sqrt(double x);

/*@ logic real S(real a, real b, real c) =
  @  \let s = (a+b+c)/2;
  @         \sqrt(s*(s-a)*(s-b)*(s-c));
  @ */

/*@ requires 0 <= c <= b <= a && a <= b + c && a <= 0x1p255;
  @ ensures 0x1p-513 < \result
  @    ==> \abs(\result-S(a,b,c)) <= (4.75*0x1p-53 + 33*0x1p-106)*S(a,b,c);
  @ */
double triangle (double a,double b, double c) {
  return (0x1p-2*sqrt((a+(b+c))*(a+(b-c))*(c+(a-b))*(c-(a-b))));
}
```

Fig. 2. Annotated and Proved C program for the computation of the area of a triangle

function with a specification: it requires the input to be non negative and produces the rounding to nearest of the exact square root. Then S is the mathematical exact value of the area of the triangle, computed with Heron's formula. We then require the inputs of the function are such that $0 \leq c \leq b \leq a \leq b + c$ as explained before. Note that in the annotations, all computations are exact, therefore the addition $b+c$ is the mathematical addition.

We also require that $a \leq 2^{255}$. The reason is to prevent overflows. This is sufficient to guarantee that no operation will overflow. It may seem a strong hypothesis, but if you consider $a = b = c = 2^{256}$, then M is the rounding of $3 \cdot 2^{1024}$ and thus overflows.

The last annotation describes what the function ensures, *i.e.* guarantees: if the result is greater than 2^{-513}, then the relative error is smaller than $4.75{\cdot}2^{-53}{+}33{\cdot}2^{-106}$.

Let us now detail the proofs. There are 3 kinds of proofs. The first one is the precondition of the square root function that requires the input to be non negative. This was already proved in Coq and we just had to plug the given proof. The second kind concerns the overflows. All those were automatic thanks to Gappa: the hypothesis $a \leq 2^{255}$ and the facts that
$0 \leq c \leq b \leq a$ were sufficient for Gappa to prove no exceptional behavior (infinities here) will occur.

The last proof is the one of what the function ensures. First we wanted to compare our algorithm to Heron's formula, so we first prove that, for all real numbers a, b and c, if $s = \frac{a+b+c}{2}$, then $\sqrt{s\ (s-a)\ (s-b)\ (s-c)} = \frac{1}{4}\sqrt{(a+(b+c))\ (c-(a-b))\ (c+(a-b))\ (a+(b-c))}$. The proof is straightforward using the `ring` tactic.

For the round-off error, we use the err_Δ_flt_radix2 theorem. We have several things to prove in terms of precision, minimal exponent, and so on. The hypothesis $2^{-513} < \Delta$ is of course exactly the $2^{\lceil \frac{E_i+p-1}{2} \rceil - 2}$ required by the theorem, as $p = 53$ and $E_i = -1074$ in `binary64`.

All proof obligations were done either with Coq or with Gappa as shown in Figure 3. We proved the C program respects its specification, and that it will not fail, due to overflow or to a negative square root.

5 Conclusion

With this case study, we have shown several interesting facts. The first one is that Kahan's algorithm for computing the area of a triangle is correct, and that its error bound is much better than what Goldberg gave in [2]. The second one is that the formal proof of this fact was not as cumbersome as expected, as features from Coq were really helpful to get the error bound without much effort. Unfortunately, bounding the higher order terms was tedious and this should be automatized in the future. These proofs have also shown that the Flocq library was both complete (all needed theorems were there) and helpful. We did not find useless goals to prove or tedious stating of theorems, as can be found sometimes in the standard library of Coq.

Another fact is that taking underflow and overflow into account gives only a small range of inputs where this program is correct. As soon as a is greater than 2^{255} or if the result is smaller than 2^{-513}, we do not guarantee anything. This correct range was much smaller than what we expected. As for the proofs, overflow was very easy as it is entirely handled by Gappa. But managing possible underflows was rather complex. It lead us to reorder the computations to be sure to get a hint that a subnormal did appear. More generally, it does not seem easy to give a recipe to handle gradual underflow and each example seem to come with its own subtlety that helps to prove it, but that cannot be applied to another.

There are a few generalizations of this work. If anyone ever needs another radix than 2 or 10 or a power of 2, the proof can be re-run to get a slightly increased error bound taking into account the fact that $\frac{1}{4}$ does not fit in the format. As for the other rounding to nearest, this could probably apply to rounding to nearest, ties away from zero. The proof was unpractical with rounding to nearest with an arbitrary tie, as it causes problems with the symmetry of the rounding: we cannot prove that $\circ(|x|) = |\circ(x)|$ so we decided this complexity was not worth it and we chose ties to even.

Another perspective would be to consider the side lengths as real numbers. They will therefore come as FP numbers with an error. The previous algorithm is not safe in this case as it may lead to take the square root

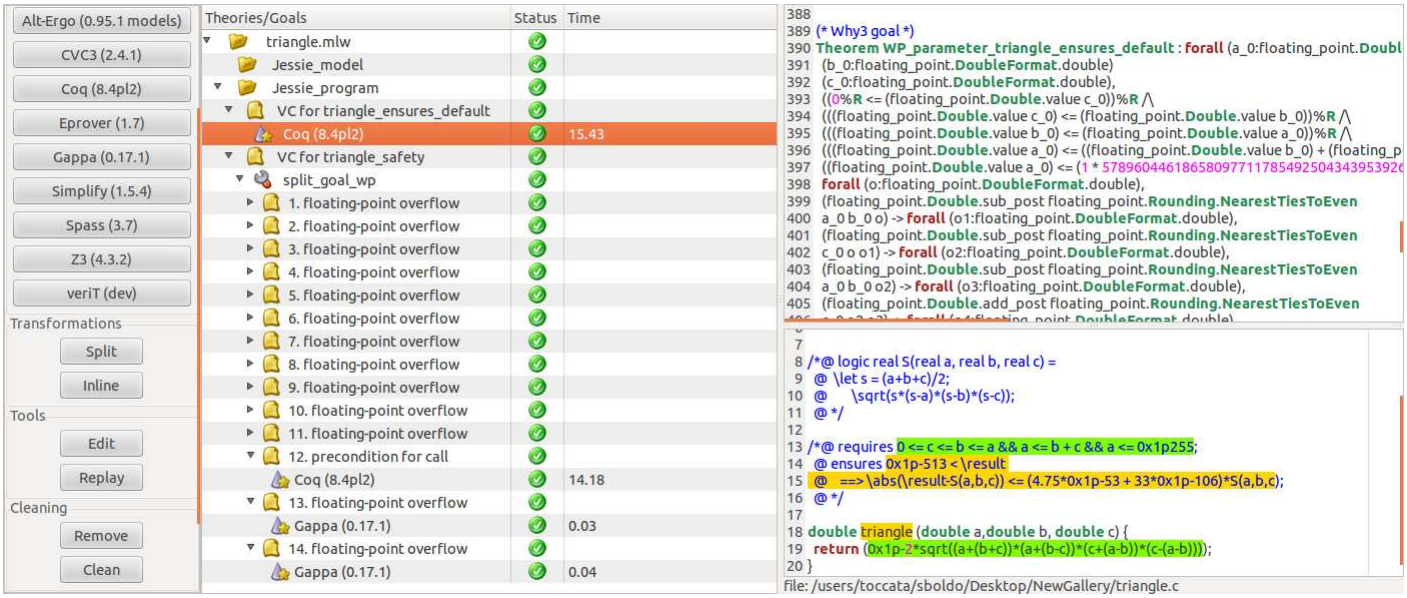

Fig. 3. Snapshot of Why3Ide: all goals were proved (green dots). The hidden goals concern overflows and are proved using Gappa. The first goal is the post-condition of the function while Goal 12 is the non-negativity of the square root (precondition of the square root function).

of a negative number. The reason is that the inequality $a \leq b + c$ will hold on the real side lengths, but not always for their roundings. Consider $a = 1 + 3 \cdot 2^{-53}$, $b = 1$ and $c = 3 \cdot 2^{-53}$. In `binary64`, the value a will be rounded in $\tilde{a} = 1 + 4 \cdot 2^{-53}$ while b and c are unchanged. Then, $\circ(c - \circ(\tilde{a} - b)) = -2^{-53}$ and the algorithm will fail. Another algorithm should be created to also handle these cases.

A long-term perspective is to consider all (or most?) the algorithms from the floating-point literature and formally prove them. It could be done under the common assumptions, that is to say no underflow and no overflow. But it would be more interesting to handle gradual underflow by either giving constraints for subnormal not to appear or by giving correct results even in this case. Overflow has also to be considered, but our experience shows that constraints on initial values are usually enough. Going from well-known facts to formally proved facts would be decisive step towards a high guarantee of our scientific results.

References

[1] W. Kahan, "Miscalculating Area and Angles of a Needle-like Triangle," 1986, Unpublished manuscript. [Online]. Available: http://www.cs.berkeley.edu/~wkahan/Triangle.pdf

[2] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–48, Mar. 1991.

[3] S. Boldo, "How to compute the area of a triangle: a formal revisit," in *Proceedings of the 21th IEEE Symposium on Computer Arithmetic*, Austin, Texas, USA, Apr. 2013, pp. 91–98.

[4] V. A. Carreño and P. S. Miner, "Specification of the IEEE-854 floating-point standard in HOL and PVS," in *HOL95: 8th International Workshop on Higher-Order Logic Theorem Proving and Its Applications*, Aspen Grove, UT, Sep. 1995.

[5] D. 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*, vol. 1, pp. 148–200, 1998.

[6] J. Harrison, "Formal verification of floating point trigonometric functions," in *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, Austin, Texas, 2000, pp. 217–233.

[7] W. Kahan, "On the Cost of Floating-Point Computation Without Extra-Precise Arithmetic," World-Wide Web document, Nov. 2004.

[8] S. Boldo, "Kahan's algorithm for a correct discriminant computation at last formally proven," *IEEE Transactions on Computers*, vol. 58, no. 2, pp. 220–225, Feb. 2009.

[9] S. Boldo and G. Melquiond, "Flocq: A unified library for proving floating-point algorithms in Coq," in *20th IEEE Symposium on Computer Arithmetic*, E. Antelo, D. Hough, and P. Ienne, Eds., Tübingen, Germany, 2011, pp. 243–252.

[10] P. H. Sterbenz, *Floating point computation*. Prentice Hall, 1974.

[11] G. Melquiond, "Proving bounds on real-valued functions with computations," in *Proceedings of the 4th international joint conference on Automated Reasoning*, ser. IJCAR '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 2–17.

[12] Microprocessor Standards Committee, "IEEE Standard for Floating-Point Arithmetic," *IEEE Std. 754-2008*, pp. 1–58, Aug. 2008.

[13] P. Baudin, P. Cuoq, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto, *ACSL: ANSI/ISO C Specification Language, version 1.5*, 2009. [Online]. Available: http://frama-c.cea.fr/acsl.html

[14] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich, "Why3: Shepherd your herd of provers," in *Boogie 2011: First International Workshop on Intermediate Verification Languages*, Wrocław, Poland, August 2011.

[15] M. Daumas and G. Melquiond, "Certification of bounds on expressions involving rounded operators," *Transactions on Mathematical Software*, vol. 37, no. 1, pp. 1–20, 2010.