How to Compute the Area of a Triangle: a Formal Revisit

Sylvie Boldo

Inria Saclay – Île-de-France, LRI, Université Paris-Sud

March 25th, 2015
Given 3 FP numbers $a$, $b$, and $c$ that are the side lengths of a triangle, we want to compute the area of this triangle.
Given 3 FP numbers $a$, $b$, and $c$ that are the side lengths of a triangle, we want to compute the area of this triangle.

The common formula is attributed to Heron of Alexandria:

$$\Delta = \sqrt{s (s - a) (s - b) (s - c)} \text{ where } s = \frac{a + b + c}{2}$$
This is known to be inaccurate using FP arithmetic in the case of needle-like triangles:

\[ \Delta = \sqrt{\frac{1}{4} \left( (a + (b + c)) \cdot (c - (a - b)) \cdot (c + (a - b)) \cdot (a + (b - c)) \right)} \]
This is known to be inaccurate using FP arithmetic in the case of needle-like triangles:

\[ a = 100000, \ b = 99999.99979, \text{ and } c = 0.00029, \]
Heron’s formula gives 17.6.
This is known to be inaccurate using FP arithmetic in the case of needle-like triangles:

- for $a = 100000$, $b = 99999.99979$, and $c = 0.00029$, Heron’s formula gives 17.6.
- for $a = 99999.9996$, $b = 99999.99994$, and $c = 0.00003$, Heron’s formula FAILS.
Motivations

- This is known to be **inaccurate** using FP arithmetic in the case of needle-like triangles:

  ![Triangle Diagram]

  \[ a \]

  \[ b \]

  \[ c \]

- for \( a = 100000, \ b = 99999.99979, \) and \( c = 0.00029, \) Heron’s formula gives 17.6.

- for \( a = 99999.9996, \ b = 99999.9994, \) and \( c = 0.00003, \) Heron’s formula **FAILS**.

- Kahan suggested sorting \( a, \ b, \ c \) and using:

  \[
  \Delta = \frac{1}{4} \sqrt{(a + (b + c)) (c - (a - b)) (c + (a - b)) (a + (b - c))}
  \]
This is known to be inaccurate using FP arithmetic in the case of needle-like triangles:

- for $a = 100000$, $b = 99999.99979$, and $c = 0.00029$, Heron’s formula gives 17.6 and Kahan’s 9.999999990.
- for $a = 99999.99996$, $b = 99999.99994$, and $c = 0.00003$, Heron’s formula FAILS and Kahan’s gives 1.118033988.

Kahan suggested sorting $a$, $b$, $c$ and using:

$$\Delta = \frac{1}{4} \sqrt{(a + (b + c)) \ (c - (a - b)) \ (c + (a - b)) \ (a + (b - c))}$$
[Kahan, Miscalculating Area and Angles of a Needle-like Triangle, 1986?]

Area $\Delta$ is accurate to within a few units in their last digits.

Proof A few facts (values are non-negative, exact subtractions) □
[Kahan, Miscalculating Area and Angles of a Needle-like Triangle, 1986?]

Area $\Delta$ is accurate to within a few units in their last digits.

Proof A few facts (values are non-negative, exact subtractions) $\square$

[Goldberg, What every computer scientist should know about floating-point arithmetic, 1991]

The rounding error of area $\Delta$ is at most $11 \varepsilon$, provided $\varepsilon < 0.005$ and subtraction and square roots are accurate.

No proof.
Motivations

- Can we formally prove the error bound? (a proof that is mechanically verified by a proof assistant)
Motivations

- Can we formally (im)-prove the error bound? 
  (a proof that is mechanically verified by a proof assistant)
Can we formally (im)-prove the error bound? (a proof that is mechanically verified by a proof assistant)

What can be said about underflows?
Motivations

- Can we formally (im)-prove the error bound? (a proof that is mechanically verified by a proof assistant)
- What can be said about underflows?
- What can be said about overflows?
Motivations

- Can we formally (im)-prove the error bound? (a proof that is mechanically verified by a proof assistant)
- What can be said about underflows?
- What can be said about overflows?
- Can we have an understandable proved program?

We will use the Coq proof assistant and the Flocq library (multi-radix, multi-format and multi-precision library).

For the formal proof of the C program, we will use the following chain: Frama-C/Jessie/Why3 (see later).
Motivations

- Can we formally (im)-prove the error bound? (a proof that is mechanically verified by a proof assistant)
- What can be said about underflows?
- What can be said about overflows?
- Can we have an understandable proved program?

- We will use the Coq proof assistant and the Flocq library (multi-radix, multi-format and multi-precision library).
Motivations

- Can we formally (im)-prove the error bound? (a proof that is mechanically verified by a proof assistant)
- What can be said about underflows?
- What can be said about overflows?
- Can we have an understandable proved program?

- We will use the Coq proof assistant and the Flocq library (multi-radix, multi-format and multi-precision library).
- For the formal proof of the C program, we will use the following chain: Frama-C/Jessie/Why3 (see later).
Side lengths: $a$, $b$ and $c$
Notations

- Side lengths: $a$, $b$ and $c$
- They are sorted and represent a (possibly degenerate) triangle:

$$0 \leq c \leq b \leq a \leq b + c$$
Notations

- Side lengths: $a$, $b$ and $c$
- They are sorted and represent a (possibly degenerate) triangle:
  \[0 \leq c \leq b \leq a \leq b + c\]
- radix $\beta$, precision $p$, machine epsilon $\varepsilon = \frac{\beta^{1-p}}{2}$. 
Notations

- Side lengths: $a$, $b$ and $c$
- They are sorted and represent a (possibly degenerate) triangle:
  
  \[ 0 \leq c \leq b \leq a \leq b + c \]
- radix $\beta$, precision $p$, machine epsilon $\varepsilon = \frac{\beta^{1-p}}{2}$.
- minimal exponent $E_i$:
  (the smallest positive (subnormal) FP number is $\beta^{E_i}$)
Notations

- Side lengths: $a$, $b$ and $c$
- They are sorted and represent a (possibly degenerate) triangle:
  $$0 \leq c \leq b \leq a \leq b + c$$
- radix $\beta$, precision $p$, machine epsilon $\varepsilon = \frac{\beta^{1-p}}{2}$.
- minimal exponent $E_i$
  (the smallest positive (subnormal) FP number is $\beta^{E_i}$)
- o rounding to nearest, ties to even
  $\oplus, \ominus, \otimes$: rounded addition, subtraction and multiplication
Notations

- Side lengths: \(a\), \(b\) and \(c\)
- They are sorted and represent a (possibly degenerate) triangle:
  \[0 \leq c \leq b \leq a \leq b + c\]
- radix \(\beta\), precision \(p\), machine epsilon \(\varepsilon = \frac{\beta^{1-p}}{2}\).
- minimal exponent \(E_i\)
  (the smallest positive (subnormal) FP number is \(\beta^{E_i}\))
- \(\circ\) rounding to nearest, ties to even
  \(\oplus, \ominus, \otimes\): rounded addition, subtraction and multiplication
- \(\mathcal{C}(e)\) the exact value of a FP expression \(e\)
  (i.e. obtained without rounding)
Notations

- Side lengths: $a$, $b$ and $c$
- They are sorted and represent a (possibly degenerate) triangle:
  \[ 0 \leq c \leq b \leq a \leq b + c \]
- radix $\beta$, precision $p$, machine epsilon $\varepsilon = \frac{\beta^{1-p}}{2}$.
- minimal exponent $E_i$:
  (the smallest positive (subnormal) FP number is $\beta^{E_i}$)
- ○ rounding to nearest, ties to even
  $\oplus, \ominus, \otimes$: rounded addition, subtraction and multiplication
- $C(e)$ the exact value of a FP expression $e$
  (i.e. obtained without rounding)
- $\text{err}(x, y, e)$ means that the relative error is less than $e$:
  \[ |x - y| \leq e \cdot |y| \]
Kahan’s algorithm

$$\Delta = \frac{1}{4} \sqrt{(a + (b + c)) \ (c - (a - b)) \ (c + (a - b)) \ (a + (b - c))}$$
Kahan’s algorithm

\[ \Delta = \frac{1}{4} \sqrt{(a + (b + c)) \, (c - (a - b)) \, (c + (a - b)) \, (a + (b - c))} \]

\[ t_1 = a \oplus (b \oplus c) \]
\[ t_2 = a \oplus (b \ominus c) \]
\[ t_3 = c \oplus (a \ominus b) \]
\[ t_4 = c \ominus (a \ominus b) \]

\[ M = t_1 \otimes t_2 \otimes t_3 \otimes t_4 \]

\[ \Delta = \circ \left( \frac{1}{4} \right) \otimes \circ (\sqrt{M}) \]
Kahan’s algorithm

\[ \Delta = \frac{1}{4} \sqrt{(a + (b + c)) \ (c - (a - b)) \ (c + (a - b)) \ (a + (b - c))} \]

\[ t_1 = a \oplus (b \oplus c) \]
\[ t_2 = a \oplus (b \ominus c) \]
\[ t_3 = c \oplus (a \ominus b) \]
\[ t_4 = c \ominus (a \ominus b) \]
\[ M = ((t_1 \otimes t_2) \otimes t_3) \otimes t_4 \]
\[ \Delta = \circ \left( \frac{1}{4} \right) \otimes \circ (\sqrt{M}) \]
Kahan’s algorithm

\[ \Delta = \frac{1}{4} \sqrt{(a + (b + c)) \ (c - (a - b)) \ (c + (a - b)) \ (a + (b - c))} \]

\[ t_1 = a \oplus (b \oplus c) \]
\[ t_2 = a \oplus (b \oplus c) \]
\[ t_3 = c \oplus (a \ominus b) \]
\[ t_4 = c \ominus (a \ominus b) \]
\[ M = (((t_1 \otimes t_2) \otimes t_3) \otimes t_4) \]
\[ \Delta = \circ \left( \frac{1}{4} \right) \otimes \circ(\sqrt{M}) \]

Let us prove the corresponding program.
1 Motivations

2 Tools for Program Verification

3 Error Bound Provided neither Underflow, nor Overflow Occur

4 Underflow Handling

5 Triangle Program Proof

6 Conclusion
The used toolchain: Frama-C/Jessie/Why

Annotated C program
The used toolchain: Frama-C/Jessie/Why

1. **Annotated C program**
2. **Frama-C/Jessie plug-in**
3. **WHY verification condition generator**
4. **Verification conditions**
The used toolchain: Frama-C/Jessie/Why

- Annotated C program
- Frama-C/Jessie plug-in
- WHY verification condition generator
- Verification conditions
- Automatic provers (Alt-Ergo, Gappa, CVC3, etc.)
- Interactive provers (Coq, PVS, etc.)
Annotation language: ACSL

- ANSI/ISO C Specification Language
Annotation language: ACSL

- ANSI/ISO C Specification Language

- *behavioral specification language* for C programs
Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
- pre-conditions and post-conditions to functions (and which variables are modified).
Annotation language: ACSL

- ANSI/ISO C Specification Language
- Behavioral specification language for C programs
- Pre-conditions and post-conditions to functions (and which variables are modified).
- Variants and invariants of the loops.

⇒ For the programmer, the specification is easy to understand.
Annotation language: ACSL

- ANSI/ISO C Specification Language
- Behavioral specification language for C programs
- Pre-conditions and post-conditions to functions (and which variables are modified).
- Variants and invariants of the loops.
- Assertions
Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
- pre-conditions and post-conditions to functions (and which variables are modified).
- variants and invariants of the loops.
- assertions
- In annotations, all computations are exact.
Annotation language: ACSL

- ANSI/ISO C Specification Language
- Behavioral specification language for C programs
- Pre-conditions and post-conditions to functions (and which variables are modified).
- Variants and invariants of the loops.
- Assertions
- In annotations, all computations are exact.

⇒ For the programmer, the specification is easy to understand.
A floating-point number is a triple:

- the **floating-point number**, really computed by the program, \( x \rightarrow x_f \) floating-point part
A floating-point number is a triple:

- the **floating-point number**, really computed by the program, 
  \( x \mapsto x_f \) floating-point part

- the **value that would have been obtained with exact computations**, 
  \( x \mapsto x_e \) exact part
A floating-point number is a triple:

- **the floating-point number**, really computed by the program, \( x \rightarrow x_f \) floating-point part
- **the value that would have been obtained with exact computations**, \( x \rightarrow x_e \) exact part
- **the value that we ideally wanted to compute** \( x \rightarrow x_m \) model part
A floating-point number is a triple:

- the floating-point number, really computed by the program, 
  \[ x \rightarrow x_f \] floating-point part  
  \[ 1+x+x\times x/2 \]

- the value that would have been obtained with exact computations, 
  \[ x \rightarrow x_e \] exact part  
  \[ 1 + x + \frac{x^2}{2} \]

- the value that we ideally wanted to compute 
  \[ x \rightarrow x_m \] model part  
  \[ \exp(x) \]
A floating-point number is a triple:

- the floating-point number, really computed by the program,
  \( x \rightarrow x_f \) floating-point part
  \[ 1 + x + \frac{x^2}{2} \]
- the value that would have been obtained with exact computations,
  \( x \rightarrow x_e \) exact part
  \[ 1 + x + \frac{x^2}{2} \]
- the value that we ideally wanted to compute
  \( x \rightarrow x_m \) model part
  \( \exp(x) \)

\( \Rightarrow \) easy to split into method error and rounding error
The proof is checked in its deep details until the computer agrees with it.

We often use formal proof checkers, meaning programs that only check a proof (they may also generate easy demonstrations).

Therefore the checker is a very short program (de Bruijn criteria: the correctness of the system as a whole depends on the correctness of a very small "kernel").
Based on the Curry-Howard isomorphism. (equivalence between proofs and $\lambda$-terms)

Few automations.

Comprehensive libraries, including on $\mathbb{Z}$ and $\mathbb{R}$.

Coq kernel mechanically checks each step of each proof.

The method is to apply successively tactics (theorem application, rewriting, simplifications...) to transform or reduce the goal down to the hypotheses.

The proof is handled starting from the conclusion.
Theorem (round\_NE\_abs)

Let $\varphi$ be a format, such that the rounding to nearest, ties to even ($\circ$) can be defined. For all $x \in \mathbb{R}$, $\circ(|x|) = |\circ(x)|$. 
Theorem (round\_NE\_abs)

Let $\phi$ be a format, such that the rounding to nearest, ties to even ($\circ$) can be defined. For all $x \in \mathbb{R}$, $\circ(|x|) = |\circ(x)|$.

Lemma round\_NE\_abs: \forall x : R,
round beta fexp ZnearestE (Rabs x) = Rabs (round beta fexp ZnearestE x).
Example of Coq theorem using the Flocq library

**Theorem (round_NE_abs)**

Let $\varphi$ be a format, such that the rounding to nearest, ties to even ($\circ$) can be defined. For all $x \in \mathbb{R}$, $\circ(|x|) = |\circ(x)|$.

Lemma round_NE_abs: forall x : R,
round beta fexp ZnearestE (Rabs x) = Rabs (round beta fexp ZnearestE x).
Proof with auto with typeclass_instances.
intros x; apply sym_eq.
unfold Rabs at 2.
destruct (Rcase abs x) as [Hx|Hx].
rewrite round_NE_opp.
apply Rabs_left1.
rewrite <- (round_0 beta fexp ZnearestE).
apply round_le...
now apply Rlt_le.
apply Rabs_pos_eq.
rewrite <- (round_0 beta fexp ZnearestE).
apply round_le...
now apply Rge_le.
Qed.

With the stating of the theorem, the Tactics, and the name of theorems.
Flocq: 16,000 lines of Coq, 700 theorems,
- any radix, any format,
- both axiomatic and computable definitions of rounding,
- effective arithmetic operators,
- numerous theorems.
More about Flocq

Flocq: 16 000 lines of Coq, 700 theorems,
- any radix, any format,
- both axiomatic and computable definitions of rounding,
- effective arithmetic operators,
- numerous theorems.

Applications:
- Frama-C/Jessie  C code certifier
- CompCert       certified C compiler

http://flocq.gforge.inria.fr/
Flocq: 16,000 lines of Coq, 700 theorems,
- any radix, any format,
- both axiomatic and computable definitions of rounding,
- effective arithmetic operators,
- numerous theorems.

Applications:
- **Frama-C/Jessie**
- **CompCert**

http://flocq.gforge.inria.fr/
Methodology

C Program

The program is correct with respect to its specifications.
Methodology

Human

Annotated C Program
(specification, invariant)

The program is correct with respect to its specifications.
The program is correct with respect to its specifications.
Methodology

Annotated C Program (specification, invariant) → Frama-C

Theorem statements

Coq ← Human

Automatic provers (Gappa)

Jessie
The program is correct with respect to its specifications.
The program is correct with respect to its specifications.
Methodology

Human → Annotated C Program (specification, invariant) → Frama-C → Jessie → Theorem statements → Coq → Human

The program is correct with respect to its specifications.
Methodology

Annotated C Program (specification, invariant) → Frama-C → Theorem statements

Human → Annotated C Program (specification, invariant)
Jessie → Theorem statements
Automatic provers (Gappa) → Theorem statements
Coq → Theorem statements

The program is correct with respect to its specifications
The program is correct with respect to its specifications.
The program is correct with respect to its specifications.
Annotated C Program (specification, invariant) → Frama-C → Theorem statements

Human → Annotated C Program

Frama-C → Theorem statements

Jessie

Automatic provers (Gappa)

Coq ← Human
Methodology

Annotated C Program (specification, invariant) → Human

Frama-C

Jessie

Theorem statements

Automatic provers (Gappa)

Coq

Human

Proved Theorems

The program is correct with respect to its specifications.
The program is correct with respect to its specifications
Plan

1 Motivations

2 Tools for Program Verification

3 Error Bound Provided neither Underflow, nor Overflow Occur

4 Underflow Handling

5 Triangle Program Proof

6 Conclusion
Set of hypotheses with unbounded exponent range:

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

- all \( t_i \) are non-negative, so the square root is innocuous
Proof sketch

- all $t_i$ are non-negative, so the square root is innocuous
- $a \ominus b = a - b$, using Sterbenz theorem
Proof sketch

- All $t_i$ are non-negative, so the square root is innocuous.
- $a \ominus b = a - b$, using Sterbenz theorem.
- $t_4 = c \ominus (a \ominus b) = c \ominus (a - b) = c - (a - b)$ because $c - (a - b)$ can be represented with the exponent of $c$ and a smaller mantissa.
Proof sketch

- all $t_i$ are non-negative, so the square root is innocuous
- $a \ominus b = a - b$, using Sterbenz theorem
- $t_4 = c \ominus (a \ominus b) = c \ominus (a - b) = c - (a - b)$, because $c - (a - b)$ can be represented with the exponent of $c$ and a smaller mantissa
- **basic forward error analysis** on the others FP additions, subtractions, and multiplications
Proof sketch

- all $t_i$ are non-negative, so the square root is innocuous
- $a \ominus b = a - b$, using Sterbenz theorem
- $t_4 = c \ominus (a \ominus b) = c \ominus (a - b) = c - (a - b)$, because $c - (a - b)$ can be represented with the exponent of $c$ and a smaller mantissa
- basic forward error analysis on the others FP additions, subtractions, and multiplications
- This lemma about the square root:
Proof sketch

- all \( t_i \) are non-negative, so the square root is innocuous
- \( a \ominus b = a - b \), using Sterbenz theorem
- \( t_4 = c \ominus (a \ominus b) = c \ominus (a - b) = c - (a - b) \), because \( c - (a - b) \) can be represented with the exponent of \( c \) and a smaller mantissa
- basic forward error analysis on the others FP additions, subtractions, and multiplications
- This lemma about the square root:


\[
\text{Theorem (err\_sqrt\_flx)}
\]

Given \( x, y, e, \) if \( 0 \leq y, \) and \( e \leq 0.5 \) and \( \text{err}(x, y, e) \), then

\[
\text{err} \left( \circ (\sqrt{x}), \sqrt{y}, \varepsilon + (1 + \varepsilon) \cdot \left( \frac{e}{2} + \frac{e^2}{4} \right) \right).
\]
Thanks to Coq inference mechanism, I did not write explicitly the rounding error of $M$ in the expected form:
Practical use of un-instantiated variables

Thanks to Coq inference mechanism, I did not write explicitly the rounding error of $M$ in the expected form:

$$
\varepsilon + (1 + \varepsilon) \cdot (\varepsilon + (1 + \varepsilon) \cdot (\varepsilon + (1 + \varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon)) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon))) \cdot \varepsilon) + 0 + (\varepsilon + (1 + \varepsilon) \cdot (\varepsilon + (1 + \varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon)) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (3/2 \cdot \varepsilon + \varepsilon \cdot \varepsilon)))) \cdot \varepsilon) \cdot 0)
$$
Theorem (\text{err}_{\Delta \text{flx}})

We have

\[
\text{err}(\Delta, \mathcal{C}(\Delta), \frac{23}{4} \varepsilon + 38 \varepsilon^2).
\]

Instead of $11 \varepsilon$, we have a formal proof of $4.75 \varepsilon ( + N \varepsilon^2 )$. If we assume radix 2, then multiplying by $1.4$ is exact:
**Theorem (err\_Δ\_flx)**

We have

\[
\text{err}(\Delta, C(\Delta), \frac{23}{4}\varepsilon + 38\varepsilon^2).
\]

Instead of \(11\varepsilon\), we have a formal proof of \(5.75\varepsilon + N\varepsilon^2\).
Final rounding error with an unbounded exponent range

Theorem (err_Δ_flx)

We have

\[ \text{err}(\Delta, \mathcal{C}(\Delta), \frac{23}{4} \varepsilon + 38 \varepsilon^2). \]

Instead of 11 \( \varepsilon \), we have a formal proof of 5.75 \( \varepsilon \) (\(+ N\varepsilon^2\)).

If we assume radix 2, then multiplying by \( \frac{1}{4} \) is exact:

Theorem (err_Δ_flx_radix2)

With \( \beta = 2 \), we have

\[ \text{err}(\Delta, \mathcal{C}(\Delta), \frac{19}{4} \varepsilon + 33 \varepsilon^2). \]

Instead of 11 \( \varepsilon \), we have a formal proof of 4.75 \( \varepsilon \) (\(+ N\varepsilon^2\)).
Plan

1. Motivations
2. Tools for Program Verification
3. Error Bound Provided neither Underflow, nor Overflow Occur
4. Underflow Handling
5. Triangle Program Proof
6. Conclusion
Assumptions

**Set of hypotheses with underflow**

- 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,
- $\varepsilon \leq \frac{1}{100}$,
- $0 \leq c \leq b \leq a \leq b + c$. 
Proof sketch

- as before, all $t_i$ are non-negative, so the square root is innocuous.
Proof sketch

- as before, all $t_i$ are non-negative, so the square root is innocuous
- as before, $a \ominus b = a - b$, using Sterbenz theorem
Proof sketch

- As before, all \( t_i \) are non-negative, so the square root is innocuous.
- As before, \( a \ominus b = a - b \), using Sterbenz theorem.
- As before, \( t_4 = c \oplus (a \ominus b) = c \oplus (a - b) = c - (a - b) \).
Proof sketch

- as before, all $t_i$ are non-negative, so the square root is innocuous
- as before, $a ⊖ b = a - b$, using Sterbenz theorem
- as before, $t_4 = c ⊖ (a ⊖ b) = c ⊖ (a - b) = c - (a - b)$
- if the result of an addition is a subnormal FP, then it is exact
Proof sketch

- as before, all $t_i$ are non-negative, so the square root is innocuous
- as before, $a \ominus b = a - b$, using Sterbenz theorem
- as before, $t_4 = c \ominus (a \ominus b) = c \ominus (a - b) = c - (a - b)$
- if the result of an addition is a subnormal FP, then it is exact
- for the square root, there is no problem, as a square root cannot be subnormal (except zero)
Proof sketch

- as before, all $t_i$ are non-negative, so the square root is innocuous
- as before, $a \ominus b = a - b$, using Sterbenz theorem
- as before, $t_4 = c \oplus (a \ominus b) = c \ominus (a - b) = c - (a - b)$
- if the result of an addition is a subnormal FP, then it is exact
- for the square root, there is no problem, as a square root cannot be subnormal (except zero)
- for multiplication, we have to assume the result is not subnormal:
Proof sketch

- as before, all $t_i$ are non-negative, so the square root is innocuous
- as before, $a \ominus b = a - b$, using Sterbenz theorem
- as before, $t_4 = c \ominus (a \ominus b) = c \ominus (a - b) = c - (a - b)$
- if the result of an addition is a subnormal FP, then it is exact
- for the square root, there is no problem, as a square root cannot be subnormal (except zero)
- for multiplication, we have to assume the result is not subnormal:

**Theorem (err_mult_flt)**

Given $x_1, y_1, e_1, x_2, y_2, e_2$, if $x_1$ and $x_2$ fit in the format, if err($x_1, y_1, e_1$), and err($x_2, y_2, e_2$), and if $\beta^{E_i+p-1} < |x_1 \otimes x_2|$, then

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

- We need to prove no subnormal is created.
Detect subnormals

- We need to prove no subnormal is created.
  ⇒ detect afterwards if a subnormal appeared in the computation
Detect subnormals

- We need to prove no subnormal is created.
  ⇒ detect afterwards if a subnormal appeared in the computation
- If the result is big enough, it will mean no underflow happened anywhere in the computation.
Detect subnormals

- We need to prove no subnormal is created.
  ⇒ detect afterwards if a subnormal appeared in the computation
- If the result is big enough, it will mean no underflow happened anywhere in the computation.
  ⇒ order the $t_i$s by magnitude.
Detect subnormals

- We need to prove no subnormal is created.
  ⇒ detect afterwards if a subnormal appeared in the computation
- If the result is big enough, it will mean no underflow happened anywhere in the computation.
  ⇒ order the $t_i$s by magnitude.

**Theorem**

We have $0 \leq t_4 \leq t_3 \leq t_2 \leq t_1$. 

Detect subnormals

- We need to prove no subnormal is created.
  - detect afterwards if a subnormal appeared in the computation
- If the result is big enough, it will mean no underflow happened anywhere in the computation.
  - order the $t_i$s by magnitude.

**Theorem**

*We have $0 \leq t_4 \leq t_3 \leq t_2 \leq t_1$.***

- use the following fact to “not lose” subnormality:
Detect subnormals

- We need to prove no subnormal is created.
  ⇒ detect afterwards if a subnormal appeared in the computation
- If the result is big enough, it will mean no underflow happened anywhere in the computation.
  ⇒ order the $t_i$s by magnitude.

**Theorem**

*We have $0 \leq t_4 \leq t_3 \leq t_2 \leq t_1$.***

⇒ use the following fact to “not loose” subnormality:

**Theorem (subnormal_aux)**

*Given $x$ and $y$, we assume that $x$ is a FP and that $\beta^{E_i+p-1} < |x \otimes y|$. We also assume that, if $|x| \leq 1$, then $|y| \leq 1$. Then $\beta^{E_i+p-1} < |x|$.***
Theorem (err_Δ_flt)

We assume that \( \frac{1}{4}\beta \left\lceil \frac{E_i + p - 1}{2} \right\rceil < \Delta \). We have

\[
\text{err}(\Delta, C(\Delta), \frac{23}{4}\varepsilon + 38\varepsilon^2).
\]
Theorem (err_\Delta_flt)

We assume that \( \frac{1}{4} \beta \left\lceil \frac{E_i + p - 1}{2} \right\rceil < \Delta \). We have

\[
\text{err}(\Delta, \mathcal{C}(\Delta), \frac{23}{4} \varepsilon + 38\varepsilon^2).
\]

Theorem (err_\Delta_flt_radix2)

We assume that \( \beta = 2 \), and that \( 2 \left\lceil \frac{E_i + p - 1}{2} \right\rceil - 2 < \Delta \). We have

\[
\text{err}(\Delta, \mathcal{C}(\Delta), \frac{19}{4} \varepsilon + 33\varepsilon^2).
\]

\( \Rightarrow \) same error bounds as before, assuming \( \Delta \) is not too small.
Overflow is only taken into account at the program level.

⇒ we will prove no overflow occur
Overflow is only taken into account at the program level.

⇒ we will prove no overflow occur

We assume $a < 2^{255}$

⇒ using Gappa, we prove no overflow occur in the 14 operations
Plan

1. Motivations
2. Tools for Program Verification
3. Error Bound Provided neither Underflow, nor Overflow Occur
4. Underflow Handling
5. Triangle Program Proof
6. Conclusion
Methodology (reminder)

C Program

Proved Theorems

The program is correct with respect to its specifications.
The program is correct with respect to its specifications
Methodology (reminder)

Human

Annotated C Program (specification, invariant) → Frama-C

Jessie → Theorem statements

The program is correct with respect to its specifications.
The program is correct with respect to its specifications.
Methodology (reminder)

Human

Annotated C Program (specification, invariant)

Frama-C

Theorem statements

Jessie

Automatic provers (Gappa)

Coq ← Human

The program is correct with respect to its specifications.
The program is correct with respect to its specifications.
The program is correct with respect to its specifications.
The program is correct with respect to its specifications.
The program is correct with respect to its specifications.
Methodology (reminder)

The program is correct with respect to its specifications.
The program is correct with respect to its specifications.
Methodology (reminder)

The program is correct with respect to its specifications.
Methodology (reminder)

The program is correct with respect to its specifications.
/*@ 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)));
/@ 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))));
}
Triangle C program

```c
/*@ 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))));
}
```

Heron's formula (no rounding)
Triangle C program

```c
/*@ 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))));
}
```

Kahan’s algorithm with properly ordered t;s
Triangle C program

```c
/*@ 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));
   @ */
```

ordered side lengths

```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);
   @ */
```

```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))));
}
```
Triangle C program

/*@ 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))));
}
/*@ requires 0 <= 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))));
}
Triangle C program

/*@ 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\cdot(s-a)\cdot(s-b)\cdot(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)))));
}
Triangle C program

```c
/*@ 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)))));
}
```
Program proof
## Triangle Area – Proof

<table>
<thead>
<tr>
<th>Proof obligations</th>
<th>Gappa</th>
<th>Coq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Coq proof</td>
<td></td>
<td>18.89</td>
</tr>
<tr>
<td><strong>Behavior</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. postcondition</td>
<td></td>
<td>16.00</td>
</tr>
<tr>
<td><strong>Safety</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. FP overflow</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>2. FP overflow</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>3. FP overflow</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>4. FP overflow</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>5. FP overflow</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>6. FP overflow</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>7. FP overflow</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>8. FP overflow</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>9. FP overflow</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>10. FP overflow</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>11. FP overflow</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>12. precondition for call</td>
<td>13.22</td>
<td></td>
</tr>
<tr>
<td>13. FP overflow</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>14. FP overflow</td>
<td>0.04</td>
<td></td>
</tr>
</tbody>
</table>
Plan

1. Motivations
2. Tools for Program Verification
3. Error Bound Provided neither Underflow, nor Overflow Occur
4. Underflow Handling
5. Triangle Program Proof
6. Conclusion
Conclusion

- tighter error bound
Conclusion

- tighter error bound
- formally proved tighter error bound
Conclusion

- tighter error bound
- formally proved tighter error bound
- rather easy formal proof (shows the library is comprehensive)
Conclusion

- tighter error bound
- formally proved tighter error bound
- rather easy formal proof (shows the library is comprehensive)
- As usual, underflow handling is tricky. Here we detect afterwards by ordering the multiplications.
We assumed \( \frac{1}{4} \) is in the format, but this assumption can be removed by increasing the error bound.
- We assumed \( \frac{1}{4} \) is in the format, but this assumption can be removed by increasing the error bound.
- We assumed rounding to nearest ties to even, but it also works for the other ties.
We assumed $\frac{1}{4}$ is in the format, but this assumption can be removed by increasing the error bound.

We assumed rounding to nearest ties to even, but it also works for the other ties.

Another accurate algorithm with inputs being inexact:

$$0 \leq c \leq b \leq a \leq b + c$$

does not imply

$$0 \leq o(c) \leq o(b) \leq o(a) \leq o(b) + o(c).$$
Perspectives

- We assumed $\frac{1}{4}$ is in the format, but this assumption can be removed by increasing the error bound.
- We assumed rounding to nearest ties to even, but it also works for the other ties.

- Another accurate algorithm with inputs being inexact:

  $$0 \leq c \leq b \leq a \leq b + c$$

  does not imply

  $$0 \leq o(c) \leq o(b) \leq o(a) \leq o(b) + o(c).$$

- A generic way to transfer proofs with unbounded exponent range into proofs with gradual underflow?