

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

Sylvie Boldo

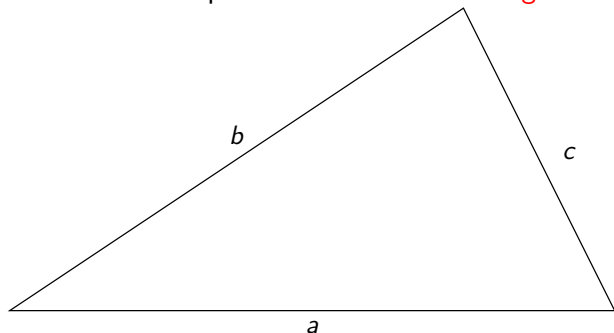
Inria Saclay – Île-de-France, LRI, Université Paris-Sud

April 9th, 2013



# Motivations

- 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**.

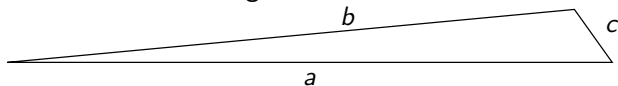


- The **common formula** is attributed to Heron of Alexandria:

$$\Delta = \sqrt{s(s-a)(s-b)(s-c)} \text{ where } s = \frac{a+b+c}{2}$$

# Motivations

- 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)  $\square$

[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 (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).

# 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
- $\mathfrak{C}(e)$  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

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

# Kahan's algorithm

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

$$t_1 = a \oplus (b \oplus c)$$

$$t_2 = a \oplus (b \ominus c)$$

$$t_3 = c \oplus (a \ominus b)$$

$$t_4 = c \ominus (a \ominus b)$$

$$M = t_1 \otimes t_2 \otimes t_3 \otimes t_4$$

$$\Delta = \circ \left( \frac{1}{4} \right) \otimes \circ (\sqrt{M})$$



# Kahan's algorithm

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

$$t_1 = a \oplus (b \oplus c)$$

$$t_2 = a \oplus (b \ominus c)$$

$$t_3 = c \oplus (a \ominus b)$$

$$t_4 = c \ominus (a \ominus b)$$

$$M = ((t_1 \otimes t_2) \otimes t_3) \otimes t_4$$

$$\Delta = \circ \left( \frac{1}{4} \right) \otimes \circ (\sqrt{M})$$

# Plan

- 1 Motivations
- 2 Error Bound Provided neither Underflow, nor Overflow Occur
- 3 Underflow Handling
- 4 Program Proof
- 5 Conclusion

# Assumptions

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
- $a \ominus b = a - b$ , using Sterbenz theorem
- **basic forward error analysis** on the others FP additions, subtractions, and multiplications
- This lemma about the square root:

## Theorem (err\_sqrt\_flx)

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

$$\text{err}(\circ(\sqrt{x}), \sqrt{y}, \varepsilon + (1 + \varepsilon) \cdot \frac{5}{8} \cdot e).$$

## 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:

$$\begin{aligned} & \varepsilon + (1 + \varepsilon) \cdot (\varepsilon + (1 + \varepsilon) \cdot (\varepsilon + (1 + \varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + \\ & (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon))) + \varepsilon + (\varepsilon + \\ & (1 + \varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot \\ & (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon)))) \cdot \varepsilon) + \varepsilon + (\varepsilon + (1 + \varepsilon) \cdot (\varepsilon + (1 + \varepsilon) \cdot \\ & (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon))) \\ & + \varepsilon + (\varepsilon + (1 + \varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon + (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) + \\ & (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon) \cdot (2 \cdot \varepsilon + \varepsilon \cdot \varepsilon)))) \cdot \varepsilon) \end{aligned}$$

# Final rounding error with an unbounded exponent range

## Theorem (err\_Δ\_flx)

*We have*

$$\text{err}(\Delta, \mathfrak{e}(\Delta), \frac{61}{8}\varepsilon + 36\varepsilon^2).$$

Instead of  $11\varepsilon$ , we have a formal proof of  $7.625\varepsilon (+N\varepsilon^2)$ .

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

## Theorem (err\_Δ\_flx\_radix2)

*With  $\beta = 2$ , we have*

$$\text{err}(\Delta, \mathfrak{e}(\Delta), \frac{53}{8}\varepsilon + 29\varepsilon^2).$$

Instead of  $11\varepsilon$ , we have a formal proof of  $6.625\varepsilon (+N\varepsilon^2)$ .

# Plan

- 1 Motivations
- 2 Error Bound Provided neither Underflow, nor Overflow Occur
- 3 Underflow Handling**
- 4 Program Proof
- 5 Conclusion

## 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
- as before,  $a \ominus b = a - b$ , using Sterbenz theorem
- 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\_multflt)

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

# Final rounding error with underflow

## Theorem (err\_Δflt)

We assume that  $\frac{1}{4}\beta^{\lceil \frac{E_i+p-1}{2} \rceil} < \Delta$ . We have

$$\text{err}(\Delta, \mathfrak{C}(\Delta), \frac{61}{8}\varepsilon + 36\varepsilon^2).$$

## Theorem (err\_Δflt\_radix2)

We assume that  $\beta = 2$ , and that  $2^{\lceil \frac{E_i+p-1}{2} \rceil - 2} < \Delta$ . We have

$$\text{err}(\Delta, \mathfrak{C}(\Delta), \frac{53}{8}\varepsilon + 29\varepsilon^2).$$

⇒ same error bounds as before, assuming  $\Delta$  is not too small.

# What about Overflow?

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 Error Bound Provided neither Underflow, nor Overflow Occur
- 3 Underflow Handling
- 4 Program Proof**
- 5 Conclusion



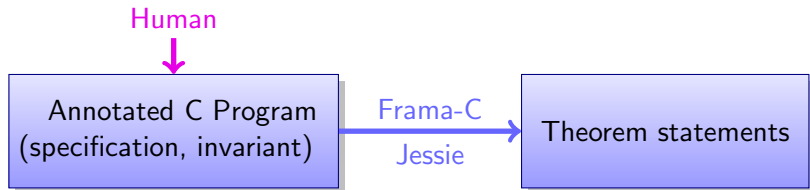
C Program

Human



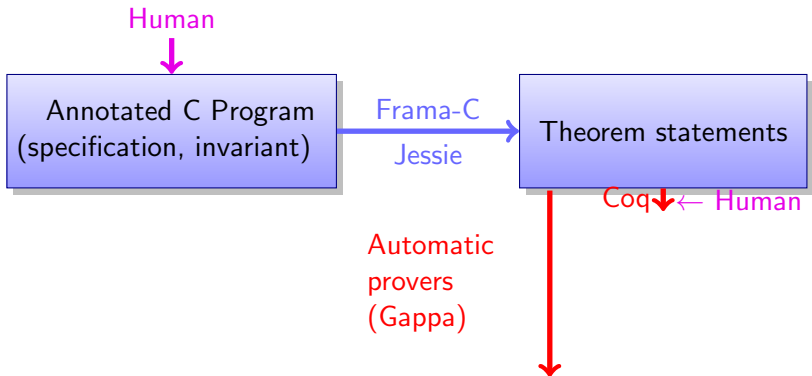
Annotated C Program  
(specification, invariant)

# Methodology

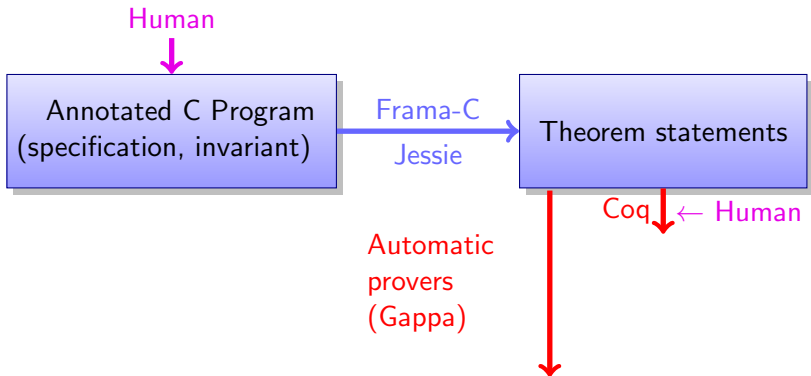




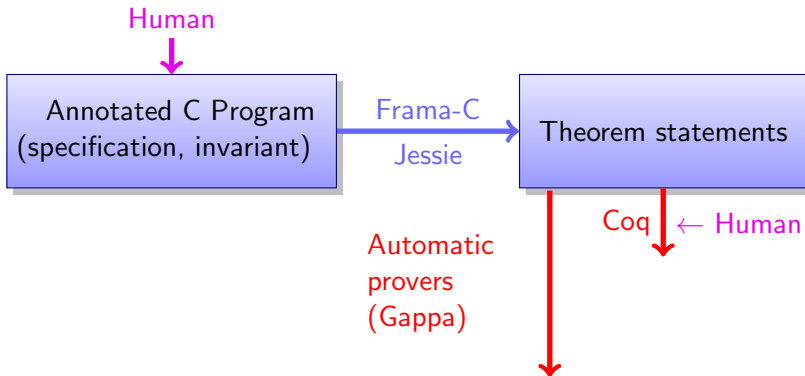
# Methodology



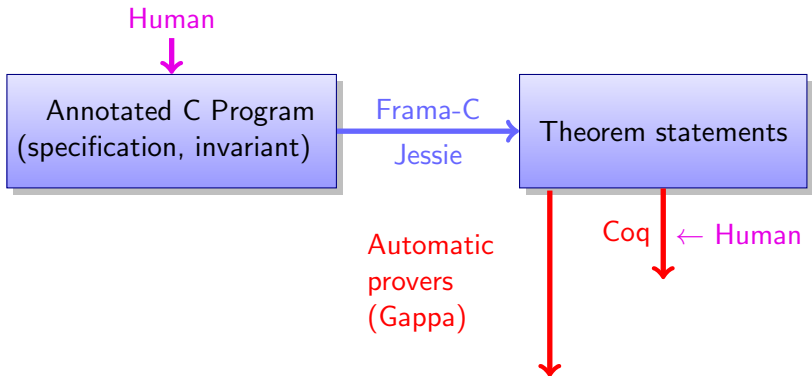
# Methodology



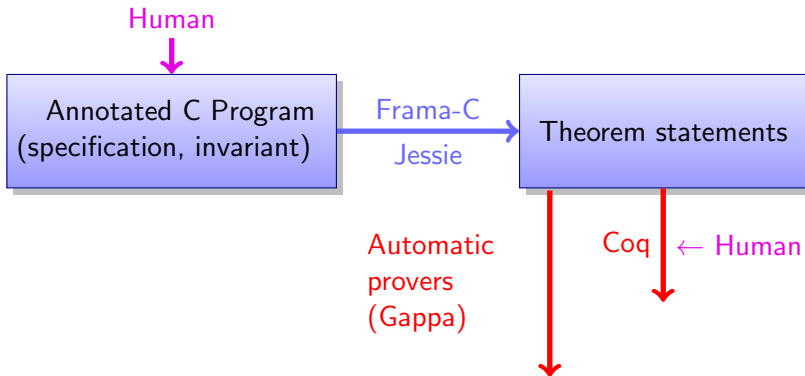
# Methodology



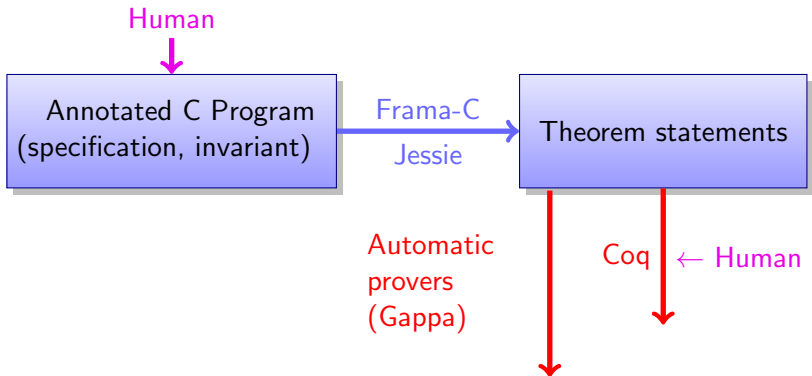
# Methodology



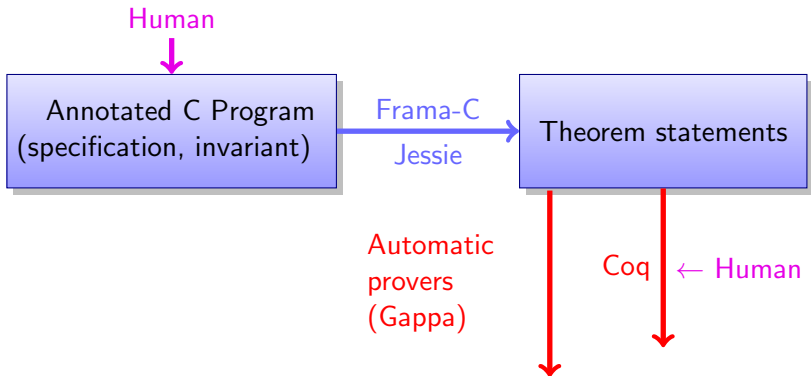
# Methodology



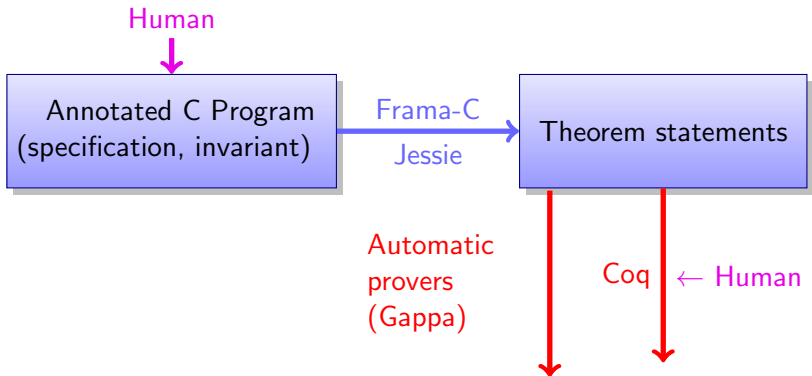
# Methodology



# Methodology

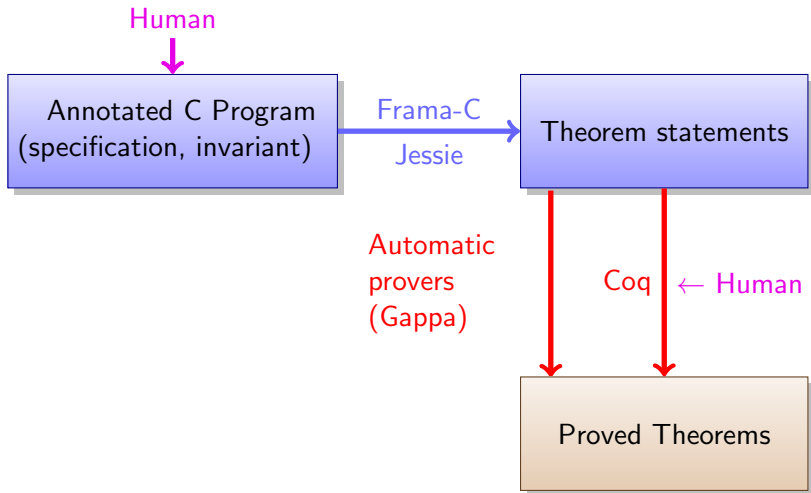


# Methodology

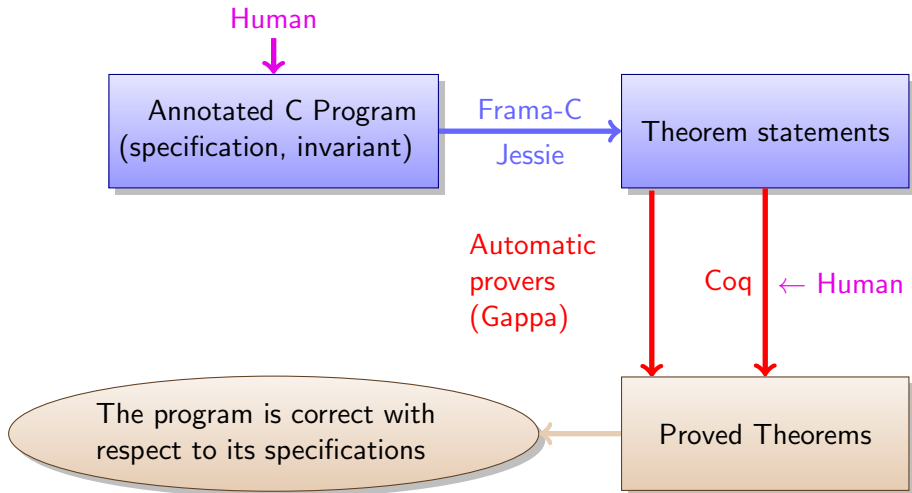




# Methodology



# Methodology



# 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))
   @ <= (53./8*0x1p-53 + 29*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);
```

Square root definition

```
/*@ 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))  
  @ <= (53./8*0x1p-53 + 29*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*(s-a)*(s-b)*(s-c));  
  @ */
```

Heron's formula  
(no rounding)

```
/*@ requires 0 <= c <= b <= a && a <= b + c && a <= 0x1p255;  
  @ ensures 0x1p-513 < \result  
  @ ==> \abs(\result-S(a,b,c))  
  @ <= (53./8*0x1p-53 + 29*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*(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))
   @ <= (53./8*0x1p-53 + 29*0x1p-106)*S(a,b,c);
   @ */
```

Kahan's algorithm  
with properly ordered  $t_i$ s

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

ordered side lengths

```
/*@ requires 0 <= c <= b <= a && a <= b + c && a <= 0x1p255;
   @ ensures 0x1p-513 < \result
   @ ==> \abs(\result-S(a,b,c))
   @ <= (53./8*0x1p-53 + 29*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*(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))
   @ <= (53./8*0x1p-53 + 29*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))));
}
```

overflow condition



# 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 If no underflow
   @ ==> \abs(\result-S(a,b,c))
   @ <= (53./8*0x1p-53 + 29*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*(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))  
  @ <= (53./8*0x1p-53 + 29*0x1p-106)*S(a,b,c);  
  @ */
```

Error bound

```
double triangle (double a, double b, double c) {  
  return (0x1p-2*sqrt((a+(b+c))*(a+(b-c))*(c+(a-b))*(c-(a-b))));  
}
```

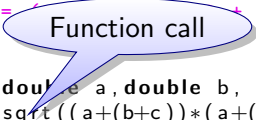
# Triangle C program

```
/*@ 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))
   @ <= 29*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

Context	Theories/Goals	Status	Time
<input checked="" type="radio"/> Unproved goals	triangle.mlw	✓	
<input type="radio"/> All goals	Jessie_model	✓	
Provers	Jessie_program	✓	
Alt-Ergo (0.93)	VC for triangle_ensures_default	✓	
CVC3 (2.4.1)	Coq (8.3pl5)	✓	9.23
Coq (8.3pl5)	VC for triangle_safety	✓	
Gappa (0.15.1)	split_goal	✓	
Z3 (3.2)	1. floating-point overflow	✓	
Transformations	2. floating-point overflow	✓	
Split	3. floating-point overflow	✓	
Inline	4. floating-point overflow	✓	
Tools	5. floating-point overflow	✓	
Edit	6. floating-point overflow	✓	
Replay	7. floating-point overflow	✓	
Cleaning	8. floating-point overflow	✓	
Remove	9. floating-point overflow	✓	
Clean	10. floating-point overflow	✓	
Proof monitoring	11. floating-point overflow	✓	
Waiting: 0	12. precondition for call	✓	
Scheduled: 0	Coq (8.3pl5)	✓	14.26
Running: 0	13. floating-point overflow	✓	
Interrupt	14. floating-point overflow	✓	

```
1734
1735 axiom H14 : of_real_post2 NearestTiesToEven 0x1.p-2 o11
1736
1737 constant o12 : double
1738
1739 axiom H15 : mul_post2 NearestTiesToEven o11 tmp o12
1740
1741 constant us_retres : double
1742
1743 axiom H16 : us_retres = o12
1744
1745 constant return : double
1746
1747 axiom H17 : return = us_retres
1748
1749 axiom H18 : 0x1.p-513 < value3 return
1750
1751 goal WP_parameter_triangle_ensures_default :
1752 abs (value3 return - us5 (value3 a_0) (value3 b_0) (value3 c_0)) <=
1753 (((53./8.0) * 0x1.p-53) + (29.0 * 0x1.p-106)) *
1754 us5 (value3 a_0) (value3 b_0) (value3 c_0))
1755 end
1756
```

```
7
8 /*@ logic real S(real a, real b, real c) =
9 @ \let s = (a+b+c)/2;
10 @ \sqrt(s*(s-a)*(s-b)*(s-c));
11 */
12
13 /*@ requires 0 <= c <= b <= a && a <= b + c && a <= 0x1p25;
14 @ ensures 0x1p-513 < \result
15 @ ==> \abs(\result-S(a,b,c)) <= (53./8*0x1p-53 + 29*0x1p-106)*S(a,b,c);
16 */
17
18 double triangle (double a, double b, double c) {
19 return (0x1p-_*sqrt(((a+(b+c))*(a+(b-c))*(c+(a-b))));
20 }
```

file: /users/demons/sboldo/Confs/ARITH13/triangle.c

# Plan

- 1 Motivations
- 2 Error Bound Provided neither Underflow, nor Overflow Occur
- 3 Underflow Handling
- 4 Program Proof
- 5 Conclusion**

# Conclusion

- tighter error bound
- **mechanically verified tighter error bound**
- rather easy formal proof (shows the library is comprehensive)
- unexpectedly tedious to handle higher-order terms in the error bounds:  $\forall \varepsilon, 0 < \varepsilon < \frac{1}{100},$

$$\varepsilon^9 + 9\varepsilon^8 + 36\varepsilon^7 + 84\varepsilon^6 + 126\varepsilon^5 + 126\varepsilon^4 + 84\varepsilon^3 + 36\varepsilon^2 + 9\varepsilon \leq 37\varepsilon^2 + 9\varepsilon$$

- As usual, **underflow handling is tricky**.  
Here we detect afterwards by ordering the multiplications.

- We assumed either ( $\beta = 2$  and  $p \geq 7$ ) or ( $\beta = 10$  and  $p \geq 3$ ). These assumptions can be removed by increasing the error bound.
- Tighter error bound:
  - $\frac{1}{2}$  instead of  $\frac{5}{8}$  in the sqrt error lemma (+ second-order terms).
  - $t_4$  is exact
- For  $\beta = 2$  and an unbounded exponent range,

$$\text{err}(\Delta, \mathfrak{C}(\Delta), 5 \varepsilon + 36\varepsilon^2).$$

- Tighter error bound **in progress**:
  - $4.75 \varepsilon$  if tightening the  $t_2$  error bound
  - same bound with underflow

Thank you for your attention!





Thank you for your attention!

