21st IEEE International Symposium on Computer Arithmetic

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

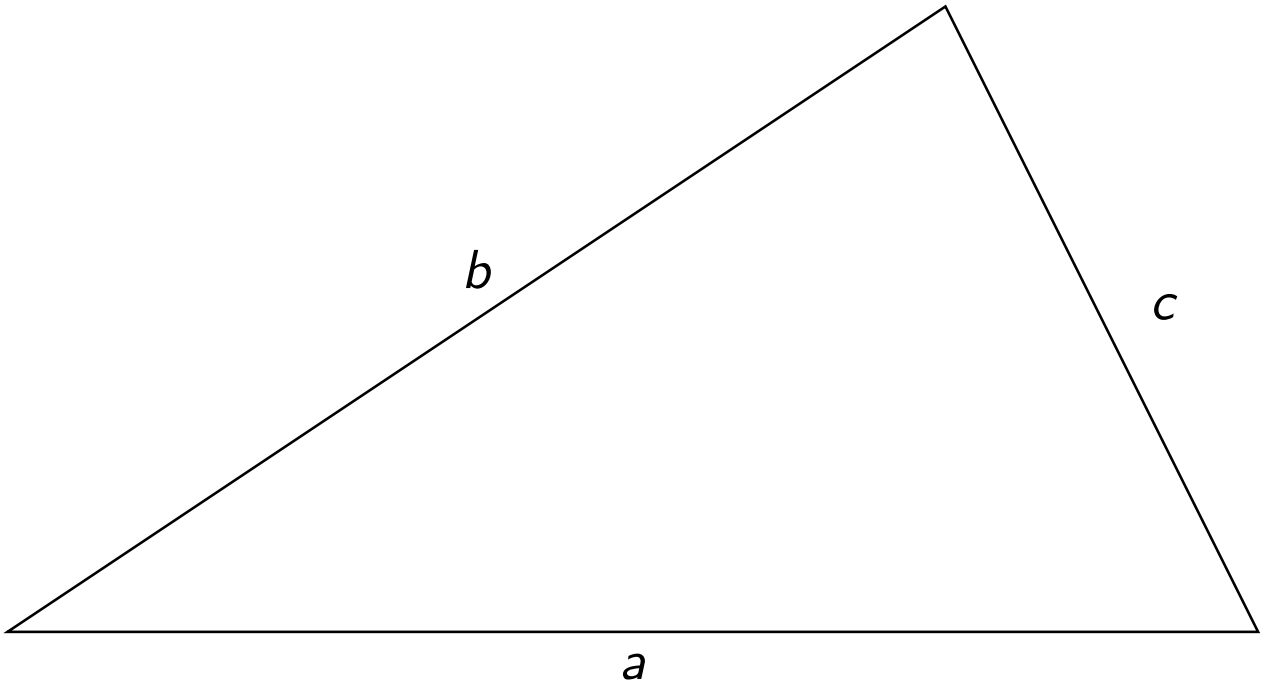
Sylvie Boldo

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.

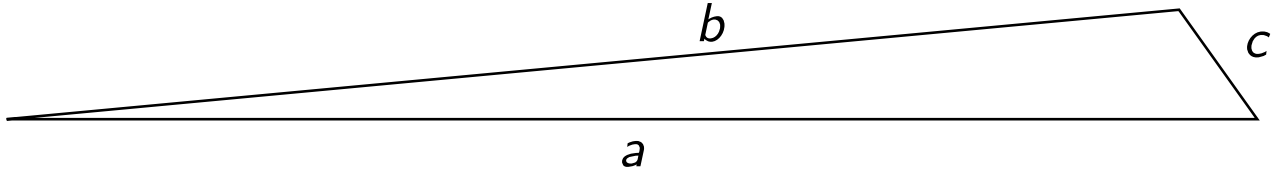


- The common formula is attributed to Heron of Alexandria:

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

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:

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

Literature

[Kahan, Miscalculating Area and Angles of a Needle-like Triangle, 1986?]

Area Δ 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 Δ is at most 11 ε, 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 β, precision p, machine epsilon $\varepsilon = \frac{\beta^{1-p}}{2}$.
- minimal exponent E_i
  (the smallest positive (subnormal) FP number is β^{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)
- $\mathrm{err}(x, y, e)$ 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

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

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

Kahan's algorithm

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

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

Plan

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* $\mathrm{err}(x, y, e)$, *then*

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

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:

$$\varepsilon + (1+\varepsilon)\cdot(\varepsilon + (1+\varepsilon)\cdot(\varepsilon + (1+\varepsilon)\cdot(2\cdot\varepsilon + \varepsilon\cdot\varepsilon + (2\cdot\varepsilon + \varepsilon\cdot\varepsilon) + (2\cdot\varepsilon + \varepsilon\cdot\varepsilon)\cdot(2\cdot\varepsilon + \varepsilon\cdot\varepsilon)) + \varepsilon + (\varepsilon + (1+\varepsilon)\cdot(2\cdot\varepsilon + \varepsilon\cdot\varepsilon + (2\cdot\varepsilon + \varepsilon\cdot\varepsilon) + (2\cdot\varepsilon + \varepsilon\cdot\varepsilon)\cdot(2\cdot\varepsilon + \varepsilon\cdot\varepsilon)))\cdot\varepsilon) + \varepsilon + (\varepsilon + (1+\varepsilon)\cdot(\varepsilon + (1+\varepsilon)\cdot(2\cdot\varepsilon + \varepsilon\cdot\varepsilon + (2\cdot\varepsilon + \varepsilon\cdot\varepsilon) + (2\cdot\varepsilon + \varepsilon\cdot\varepsilon)\cdot(2\cdot\varepsilon + \varepsilon\cdot\varepsilon)) + \varepsilon + (\varepsilon + (1+\varepsilon)\cdot(2\cdot\varepsilon + \varepsilon\cdot\varepsilon + (2\cdot\varepsilon + \varepsilon\cdot\varepsilon) + (2\cdot\varepsilon + \varepsilon\cdot\varepsilon)\cdot(2\cdot\varepsilon + \varepsilon\cdot\varepsilon)))\cdot\varepsilon))\cdot\varepsilon)$$

Final rounding error with an unbounded exponent range

Theorem (err_Δ_flx)

We have

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

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

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

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

Plan

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
- 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_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 $\mathrm{err}(x_1, y_1, e_1)$, and $\mathrm{err}(x_2, y_2, e_2)$, and if $\beta^{E_i+p-1} < |x_1 \otimes x_2|$, then

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

Detect subnormals

- We need to prove no subnormal is created.

$\Rightarrow$ 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.

$\Rightarrow$ order the t_is by magnitude.

Theorem

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

$\Rightarrow$ 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_Δ_flt)

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

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

Theorem (err_Δ_flt_radix2)

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

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

$\Rightarrow$ same error bounds as before, assuming Δ is not too small.

What about Overflow?

Overflow is only taken into account at the program level.

$\Rightarrow$ we will prove no overflow occur

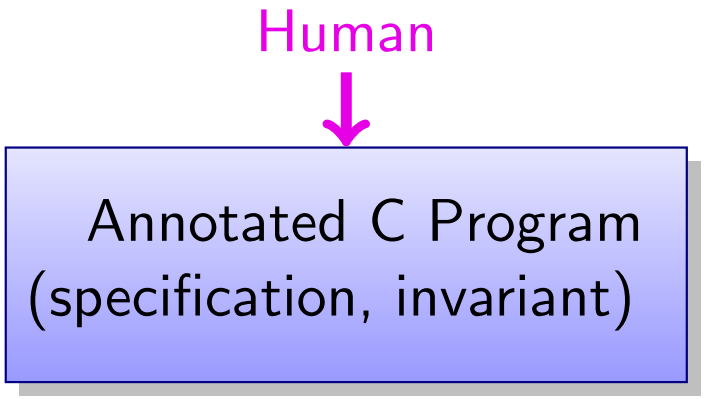
We assume $a < 2^{255}$

$\Rightarrow$ using Gappa, we prove no overflow occur in the 14 operations

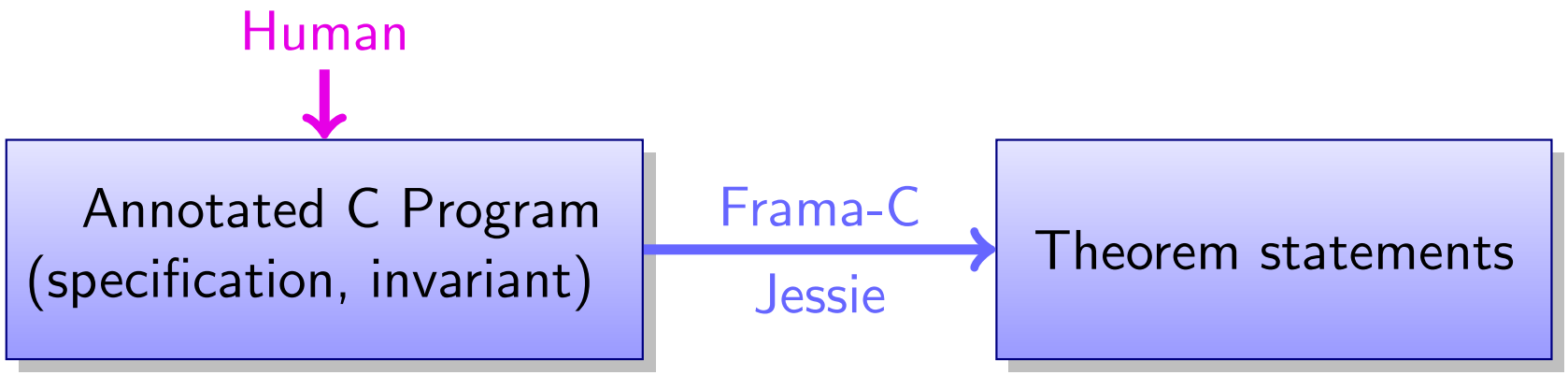
Plan

Methodology

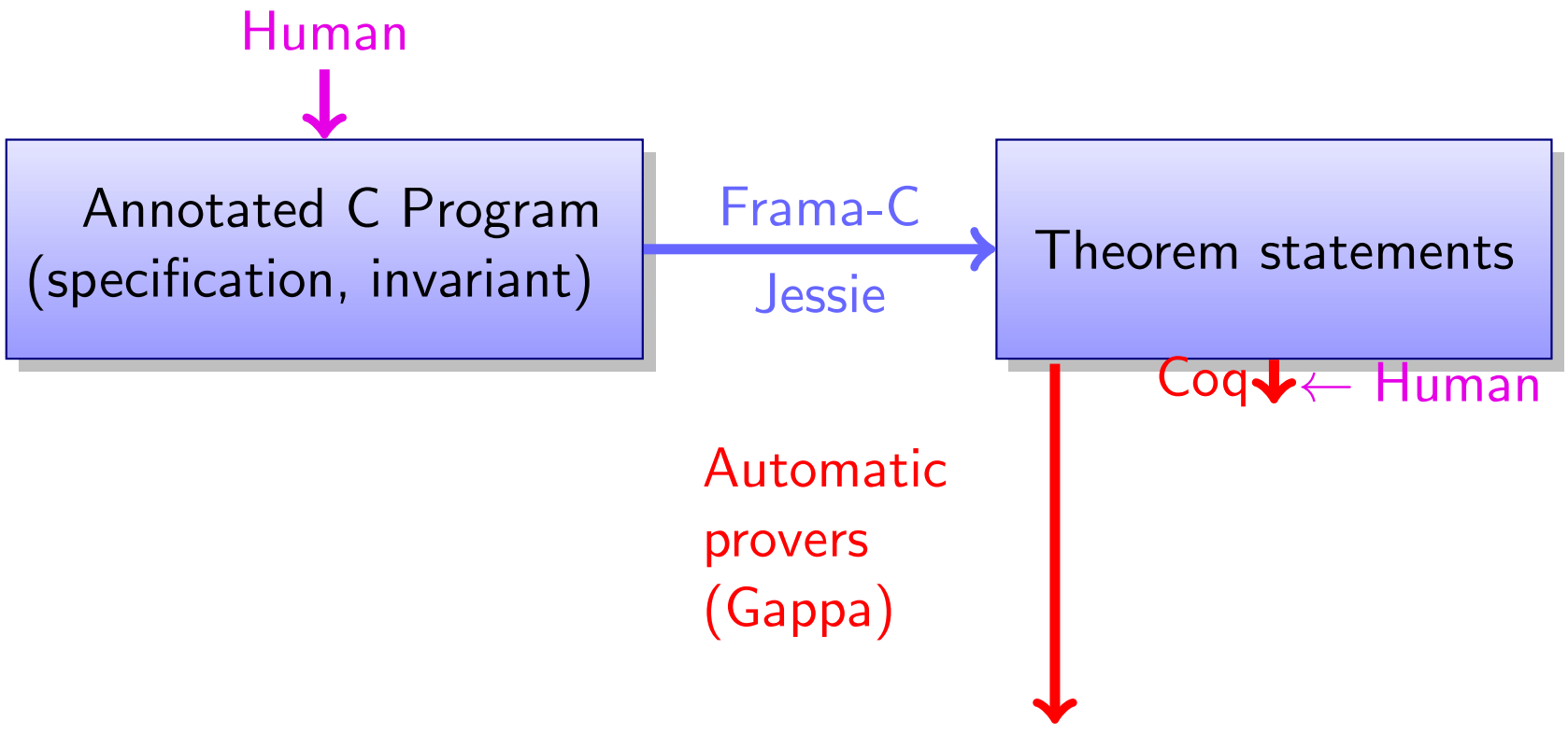
Methodology



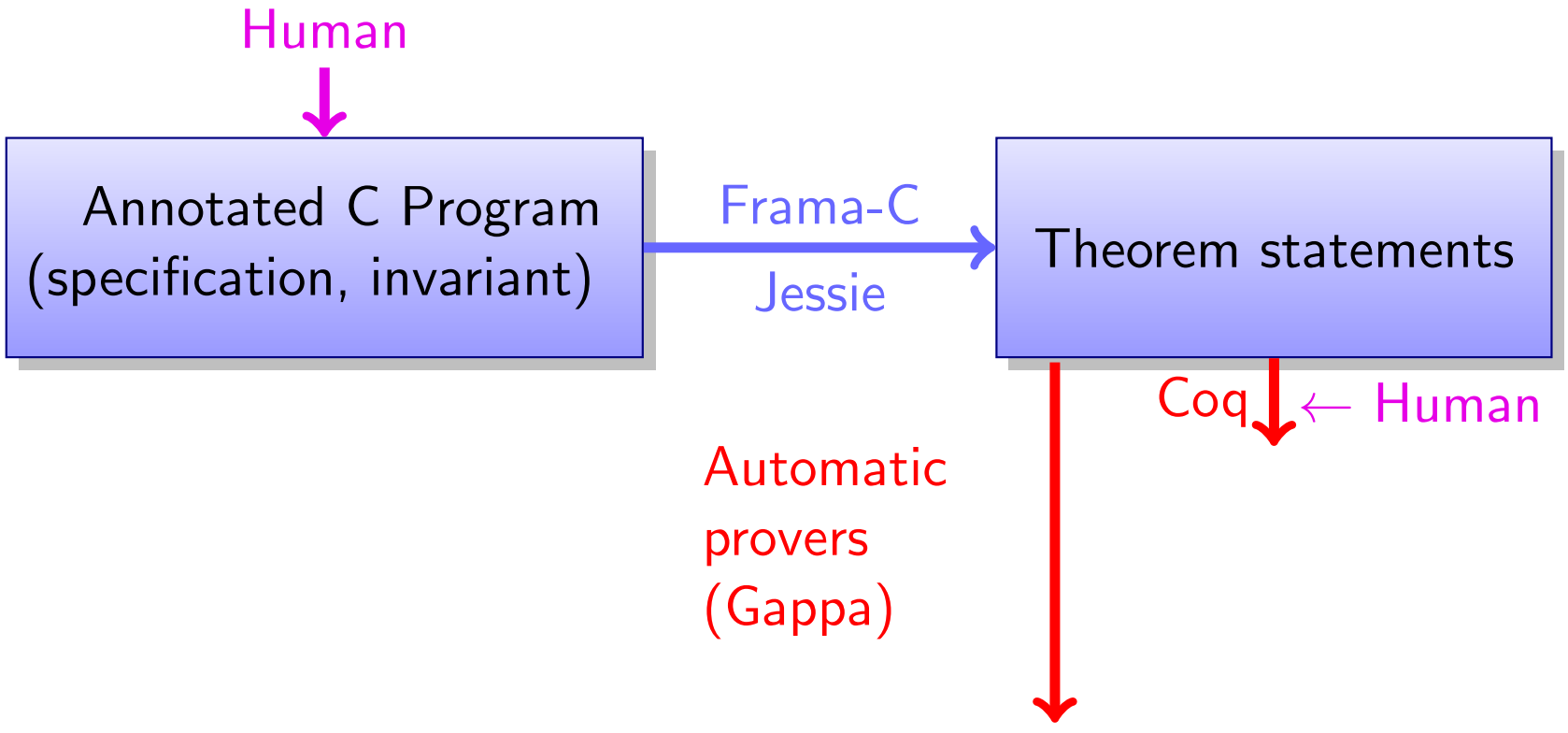
Methodology



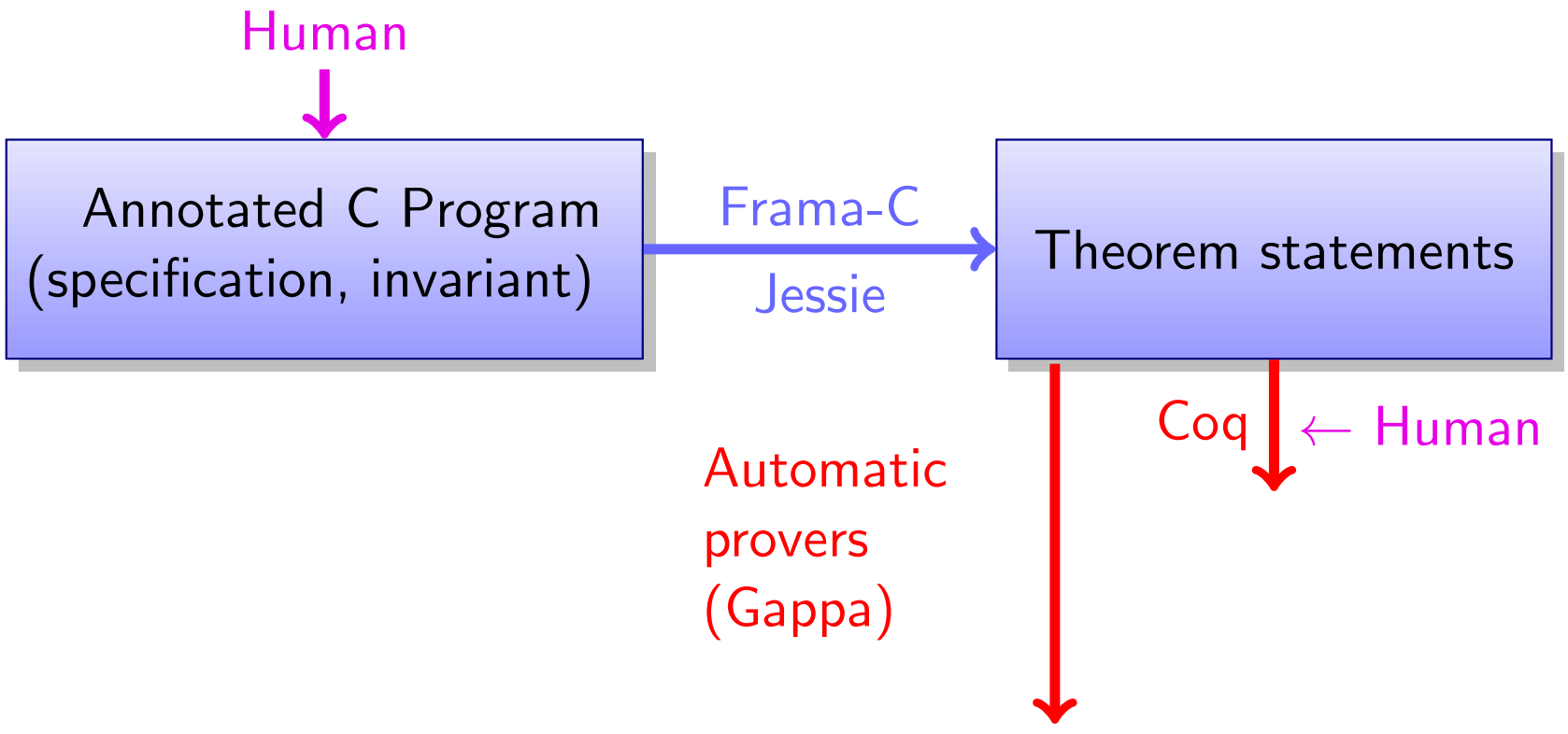
Methodology



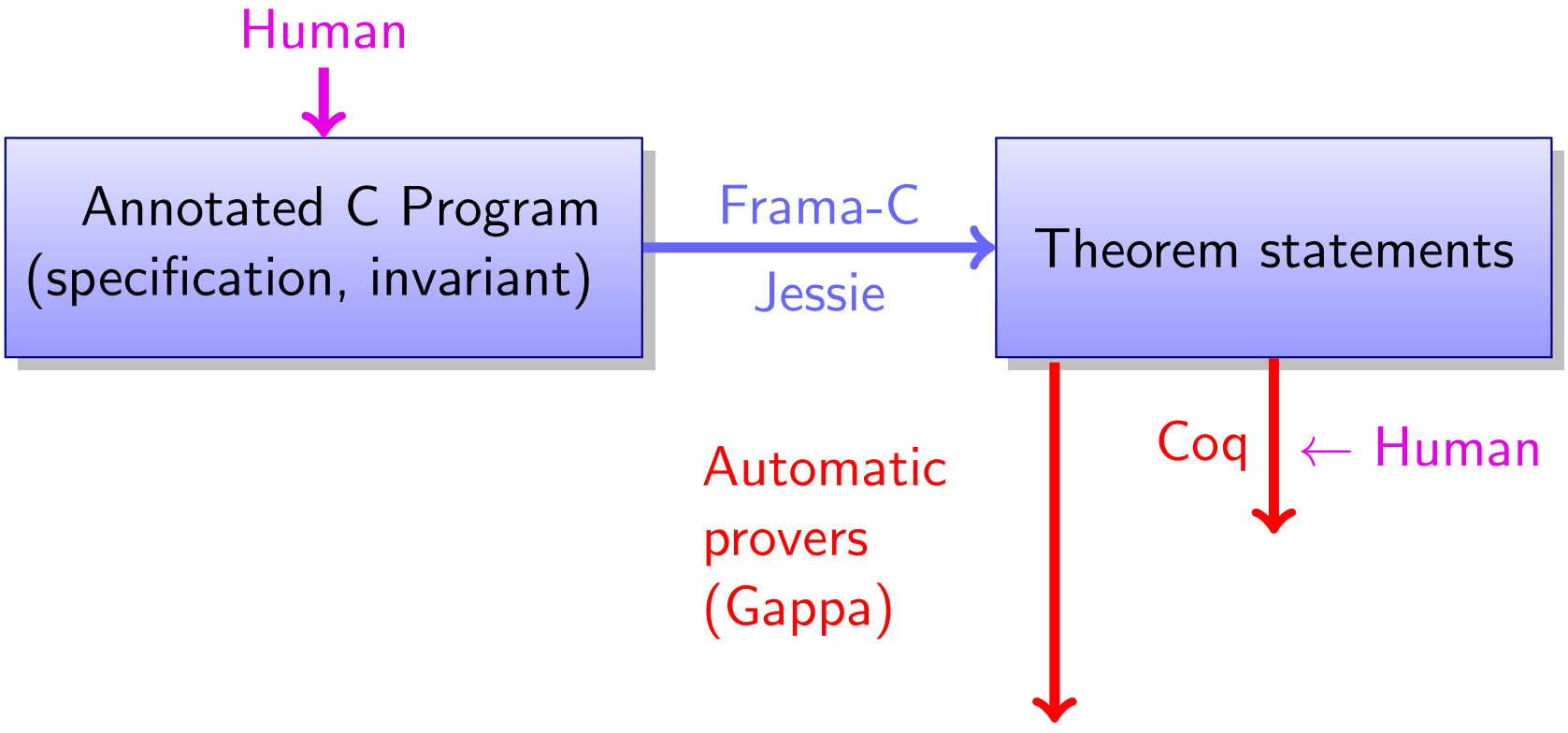
Methodology



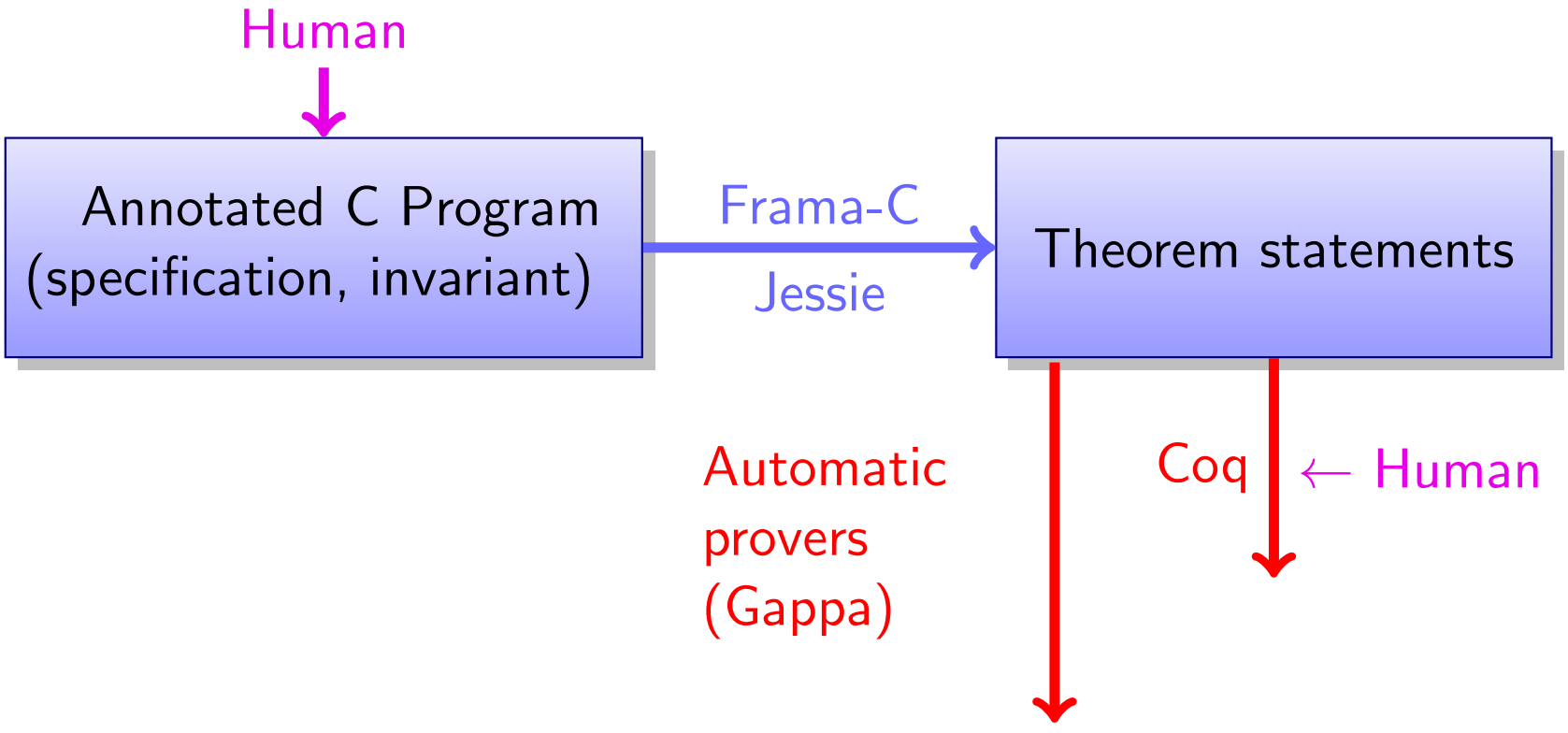
Methodology



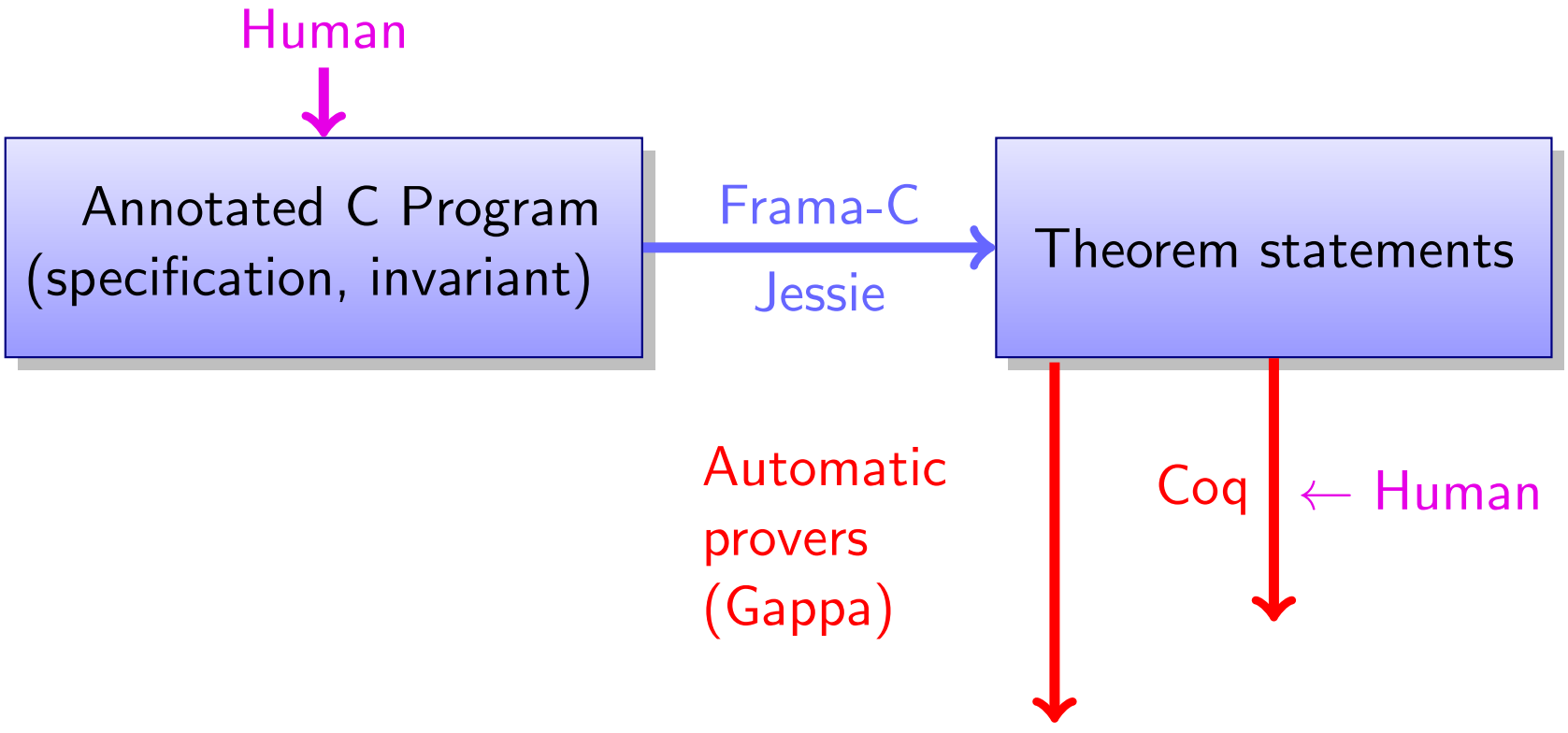
Methodology



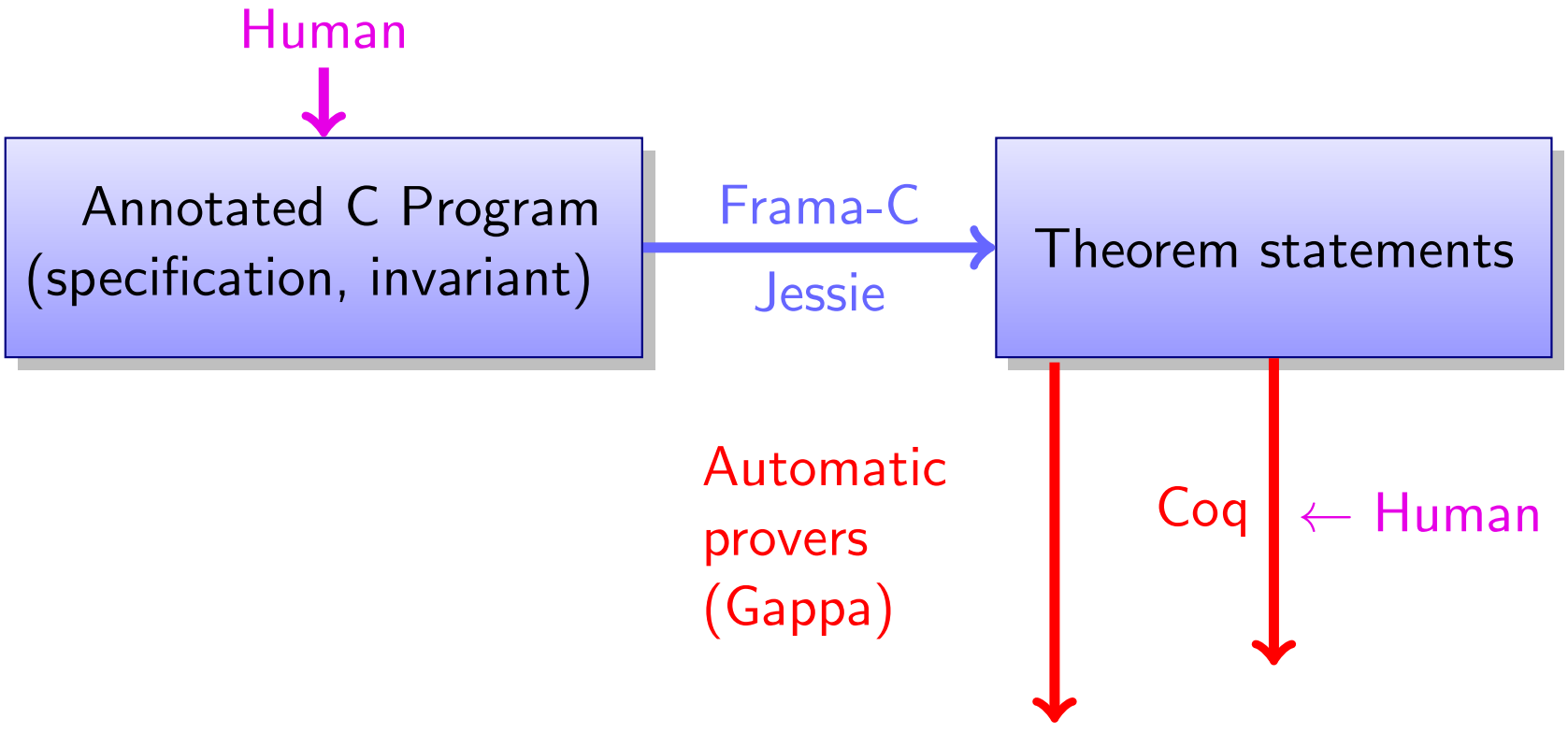
Methodology



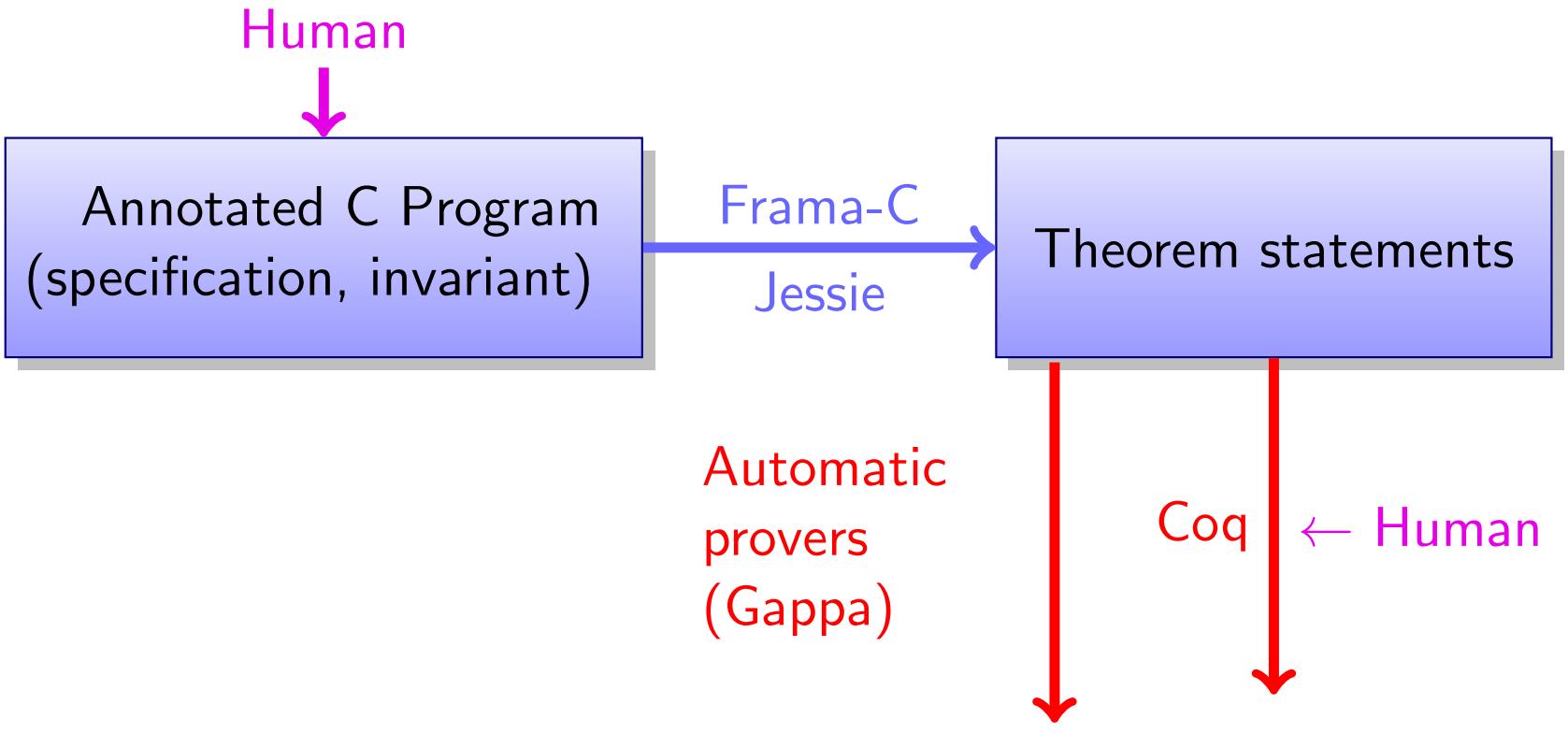
Methodology



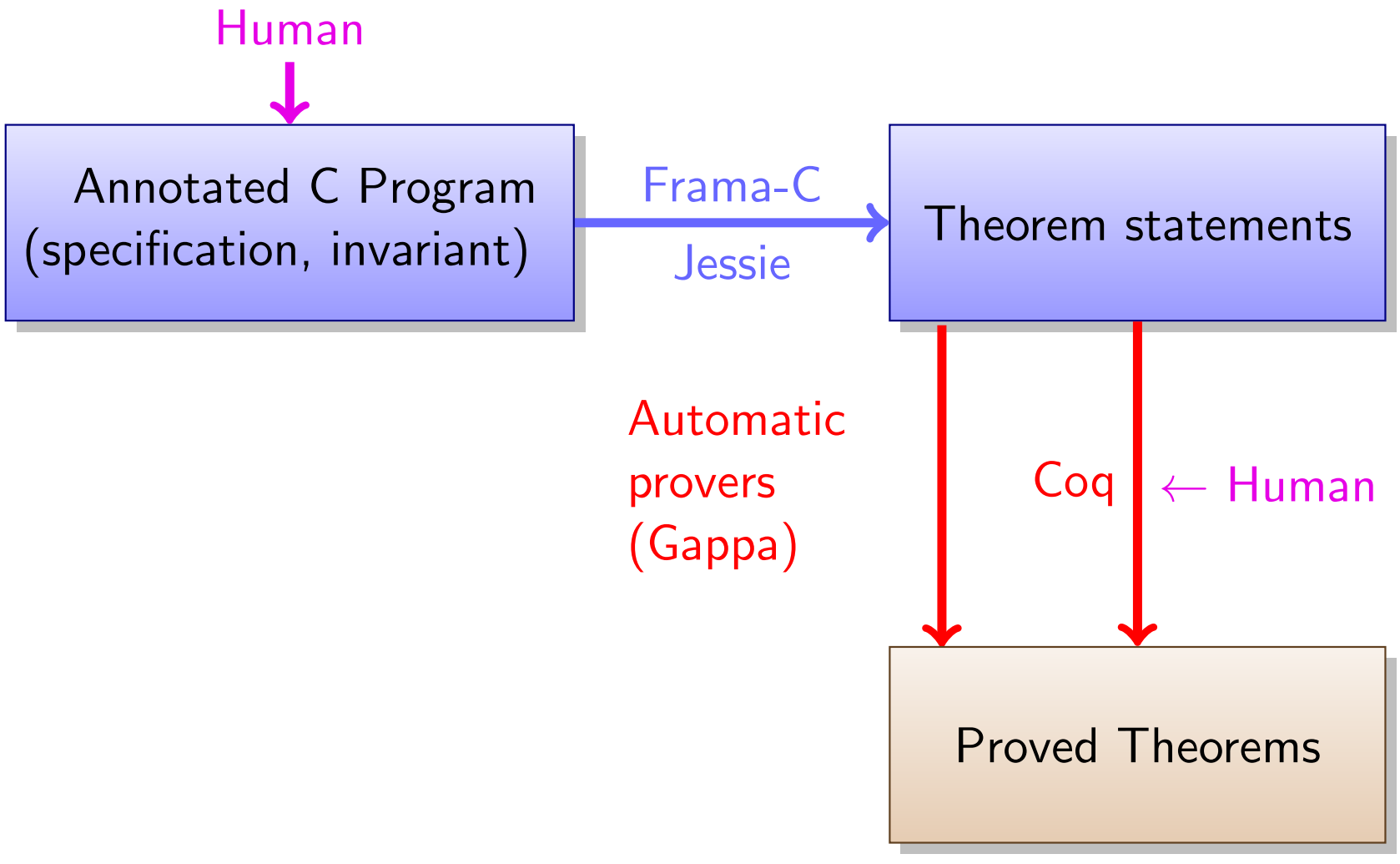
Methodology



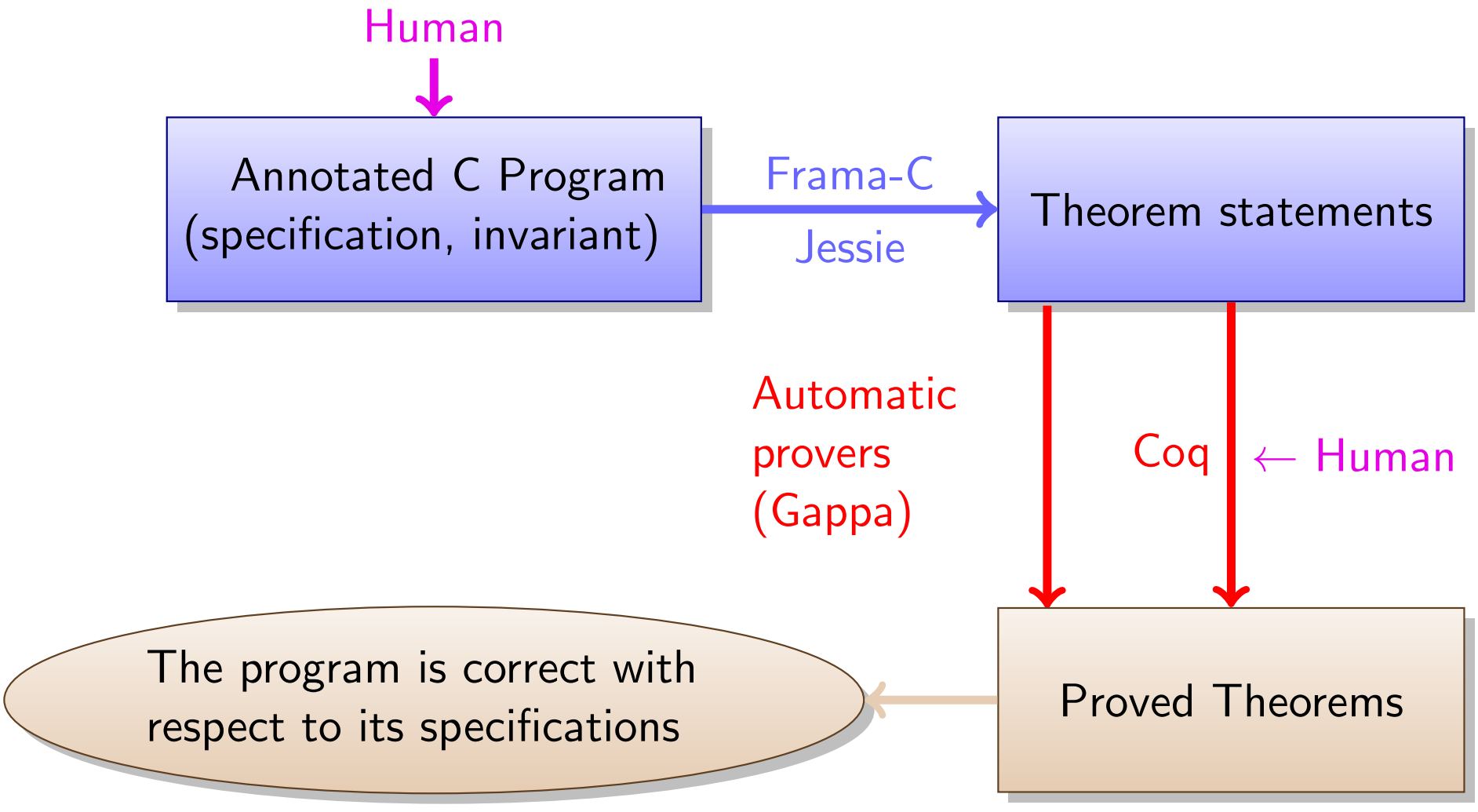
Methodology



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_is

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

overflow condition

```
/*@ 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                    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

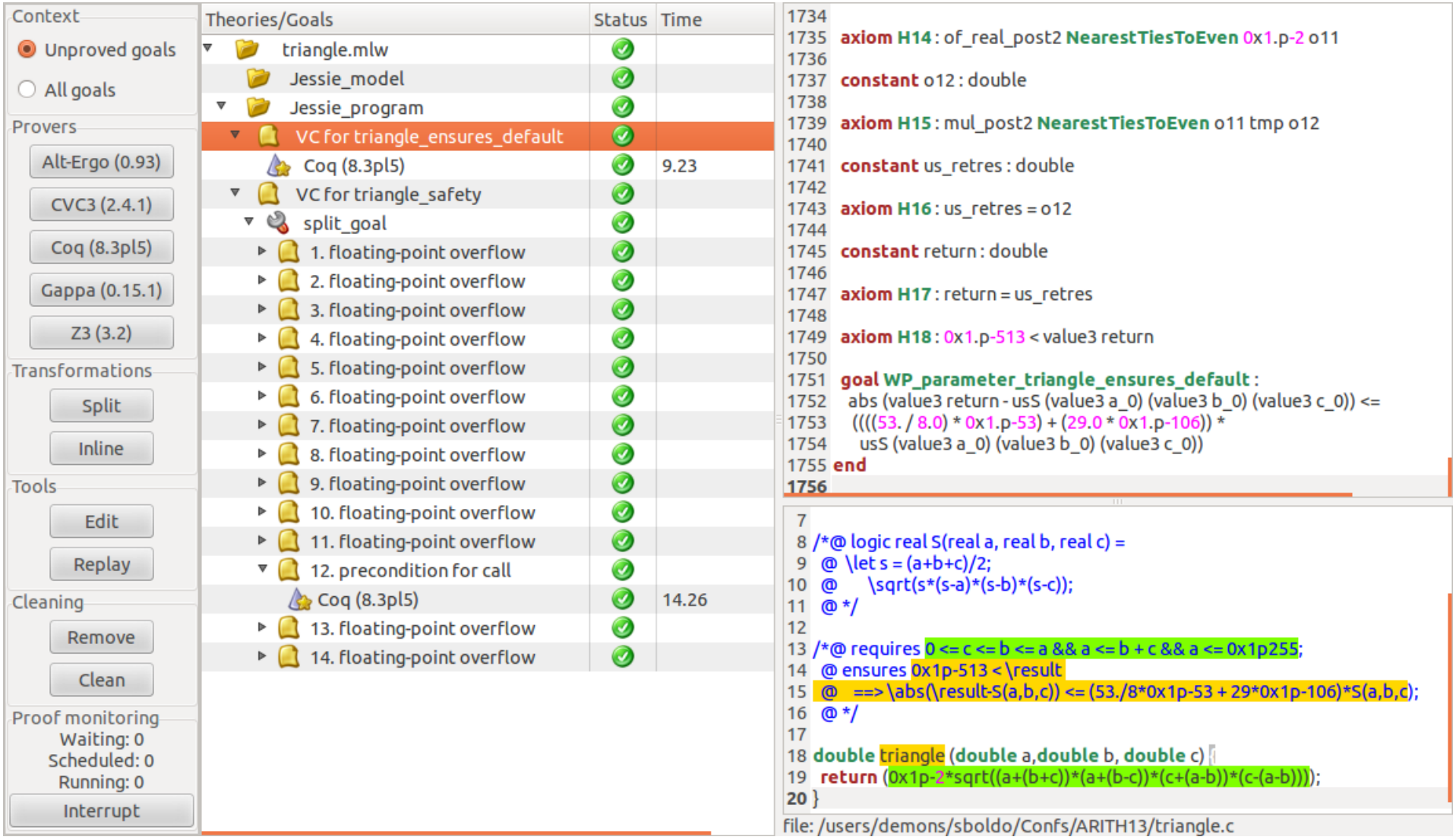
```
double triangle (double a,double b, double c) {
  return (0x1p-2*sqrt((a+(b+c))*(a+(b-c))*(c+(a-b))*(c-(a-b))));
}
```

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))));
}
```

Function call

Program proof



Plan

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.

Perspectives

- 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,

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

- Tighter error bound in progress:
  - 4.75 ε if tightening the t_2 error bound
  - same bound with underflow

Thank you for your attention!

Thank you for your attention!