Formal Verification for Numerical Computations, and the Other Way Around

Guillaume Melquiond

2019-04-01

A Bit of Vocabulary

Computer arithmetic

Art of representing numbers and performing computations on them in a mechanized fashion:

- integer arithmetic,
- fixed- and floating-point arithmetic.

A Bit of Vocabulary

Computer arithmetic

Art of representing numbers and performing computations on them in a mechanized fashion:

- integer arithmetic,
- fixed- and floating-point arithmetic.

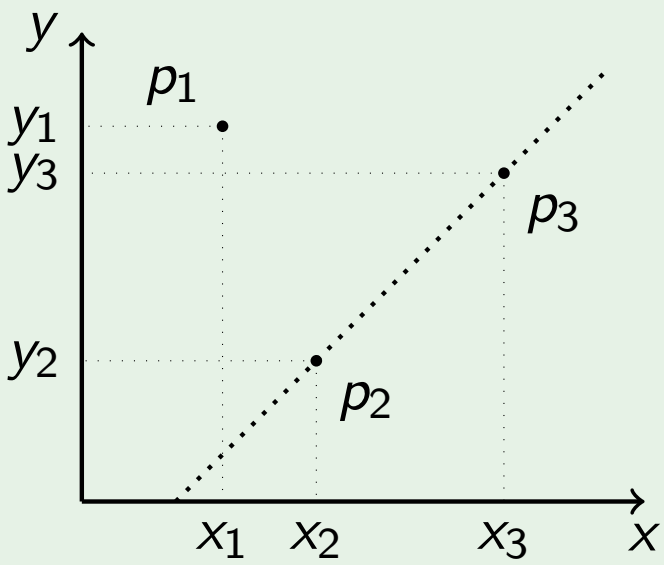
Formal verification

Art of proving the correctness of a system with respect to a formal specification, using mathematical methods.

Formal proof: well-formed formulas related by inference rules.

Orientation Detection

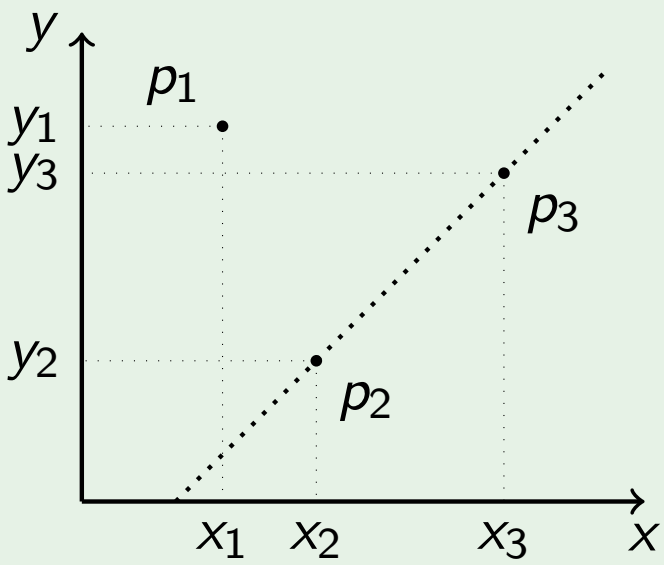
Position of p_1 with respect to $(p_2 p_3)$



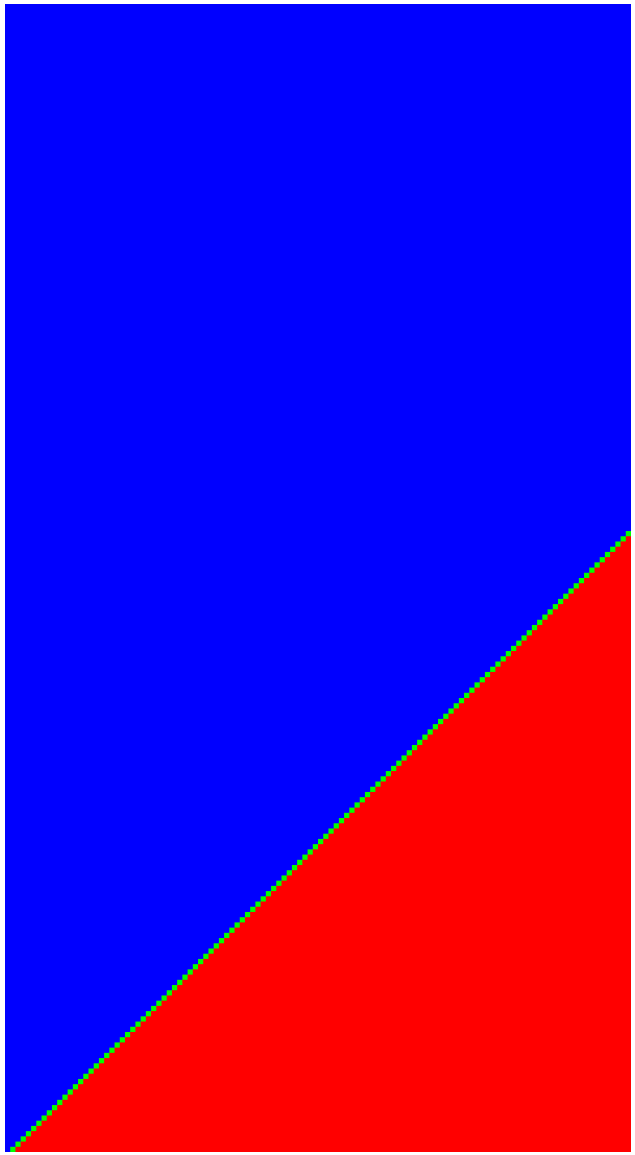
$$\text{sign} \begin{pmatrix} (x_2 - x_1) \cdot (y_3 - y_1) \\ -(y_2 - y_1) \cdot (x_3 - x_1) \end{pmatrix}$$

Orientation Detection

Position of p_1 with respect to $(p_2 p_3)$

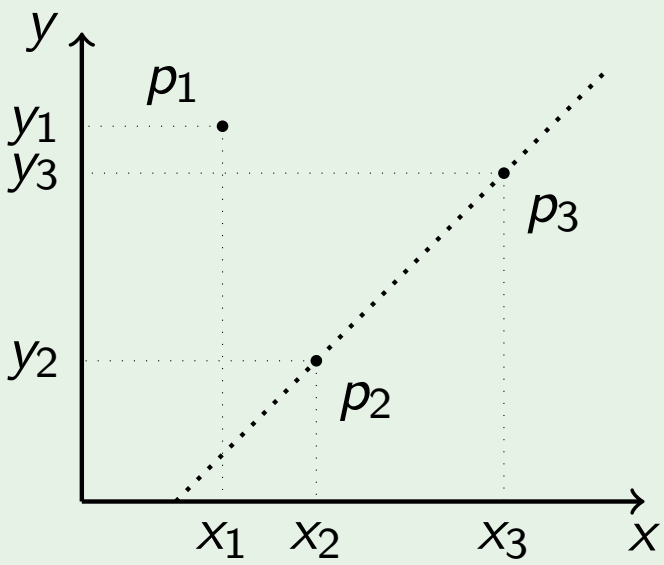


$$\text{sign}\begin{pmatrix} (x_2 - x_1)\cdot(y_3 - y_1) \\ -(y_2 - y_1)\cdot(x_3 - x_1) \end{pmatrix}$$



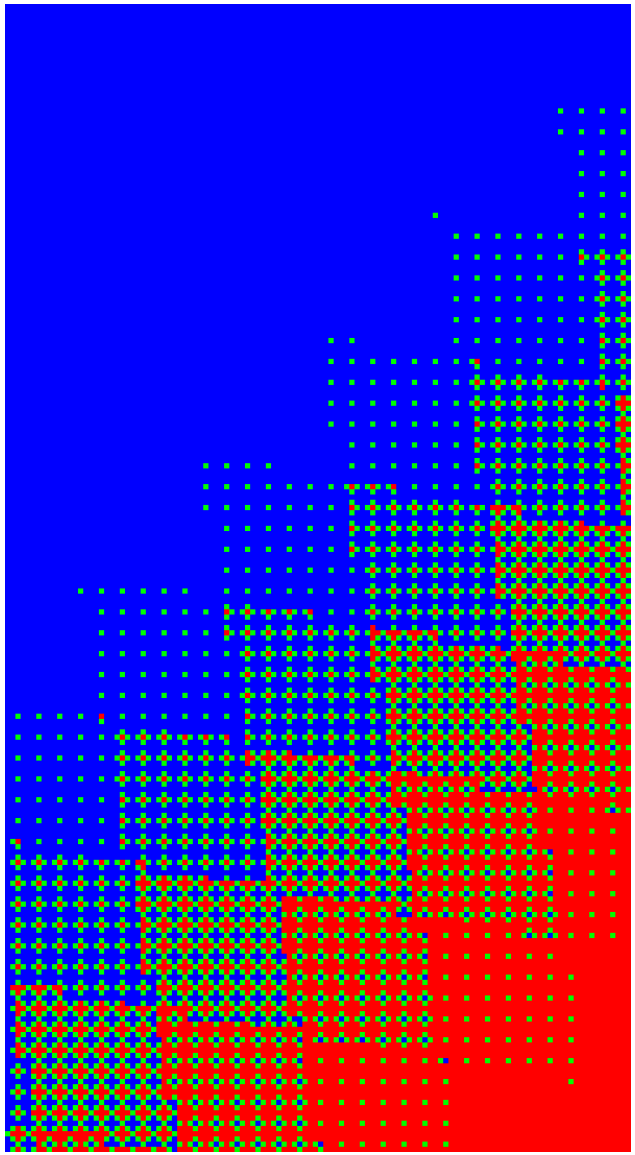
Orientation Detection

Position of p_1 with respect to $(p_2 p_3)$



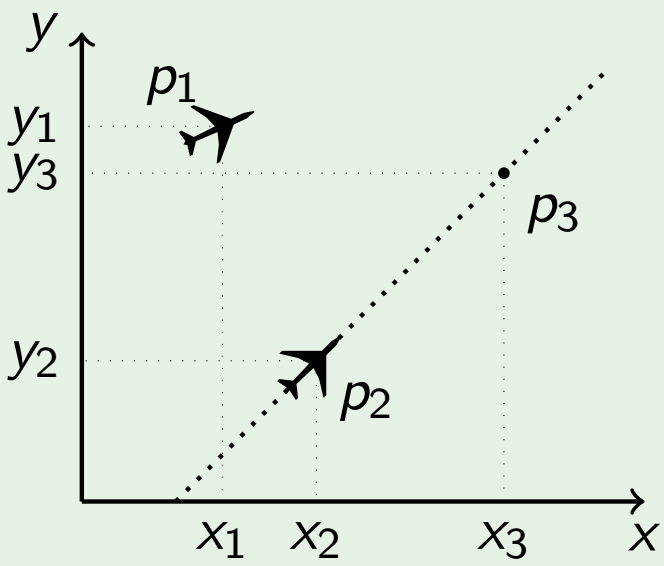
$$\text{sign} \begin{pmatrix} \circ(\circ(x_2 - x_1) \cdot \circ(y_3 - y_1)) \\ -\circ(\circ(y_2 - y_1) \cdot \circ(x_3 - x_1)) \end{pmatrix}$$

with $\circ(r) \in \mathbb{F}$ the closest floating-point number to $r \in \mathbb{R}$.



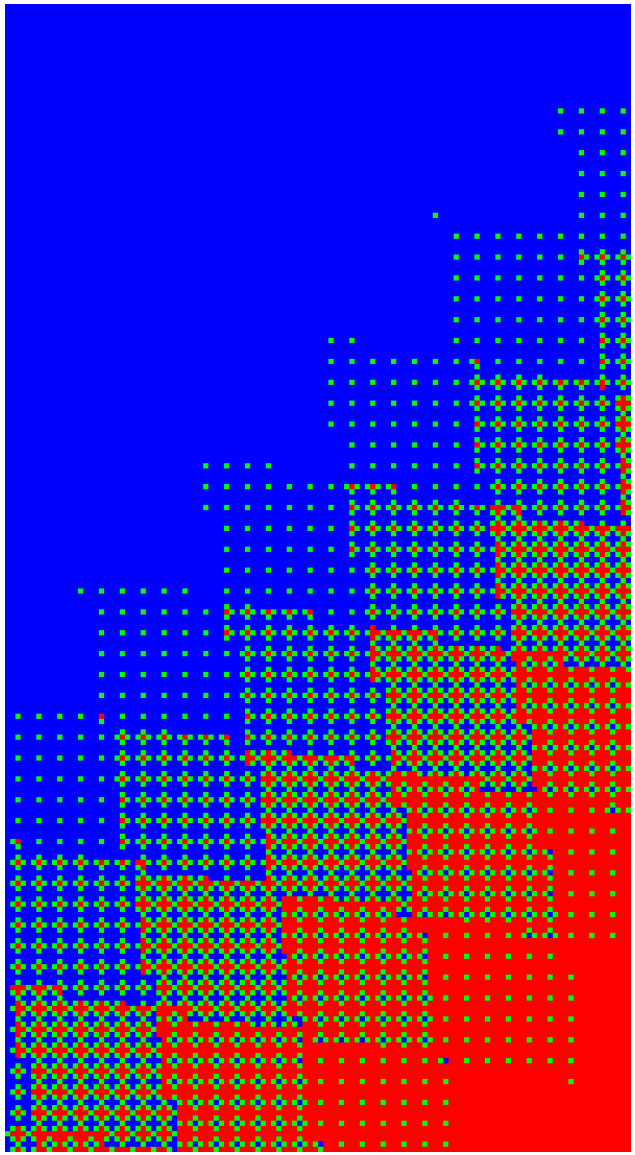
Orientation Detection

Position of p_1 with respect to $(p_2 p_3)$



$$\text{sign} \begin{pmatrix} \circ(\circ(x_2 - x_1) \cdot \circ(y_3 - y_1)) \\ -\circ(\circ(y_2 - y_1) \cdot \circ(x_3 - x_1)) \end{pmatrix}$$

with $\circ(r) \in \mathbb{F}$ the closest floating-point number to $r \in \mathbb{R}$.



Formal Verification for Numerical Computations

Situation

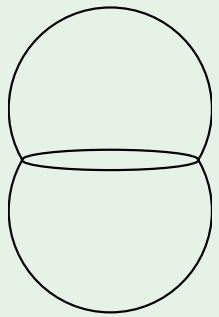
- Number representations cause subtle behaviors, if not counter-intuitive ones.
- Algorithms are more and more intricate, due to speed and space concerns.

How much can formal verification help?

Numerical Integrals in Modern Math Proofs

Double bubbles minimize (Hass, Schlafly, 2000)

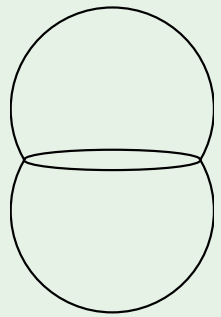
"The proof parameterizes the space of possible solutions by a two-dimensional rectangle [...]. This rectangle is subdivided into 15,016 smaller rectangles which are investigated by calculations involving a total of 51,256 numerical integrals."



Numerical Integrals in Modern Math Proofs

Double bubbles minimize (Hass, Schlafly, 2000)

"The proof parameterizes the space of possible solutions by a two-dimensional rectangle [...]. This rectangle is subdivided into 15,016 smaller rectangles which are investigated by calculations involving a total of 51,256 numerical integrals."



Major arcs for Goldbach's problem (Helfgott, 2013)

(Every odd number ≥ 7 is the sum of three prime numbers.)

$$\int_{-\infty}^{\infty} \frac{(0.5 \cdot \log(\tau^2 + 2.25) + 4.1396 + \log \pi)^2}{0.25 + \tau^2} d\tau$$

"We compute the last integral numerically (from -10^5 to 10^5)."

mathoverflow

Questions Tags Users Badges Una

Rigorous numerical integration

9

2

I need to evaluate some (one-variable) integrals that neither SAGE nor Mathematica can do symbolically. As far as I can tell, I have two options:

(a) Use GSL (via SAGE), Maxima or Mathematica to do numerical integration. This is really a non-option, since, if I understand correctly, the "error bound" they give is not really a guarantee.

(b) Cobble together my own programs using the trapezoidal rule, Simpson's rule, etc., and get rigorous error bounds using bounds I have for the second (or fourth, or what have you) derivative of the function I am integrating. This is what I have been doing.

Is there a third option? Is there standard software that does (b) for me?

na.numerical-analysis

share cite improve this question

asked Mar 5 '13 at 23:03

H A Helfgott

3,141 • 17 • 61

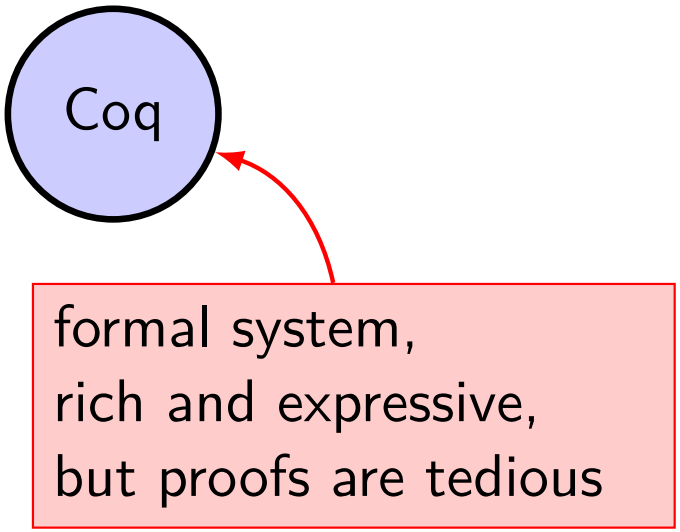
Numerical Computations for Formal Verification

- Modern mathematical proofs rely more and more on computers, by sheer necessity.
- There is a lingering doubt: what if one of the computations went wrong?

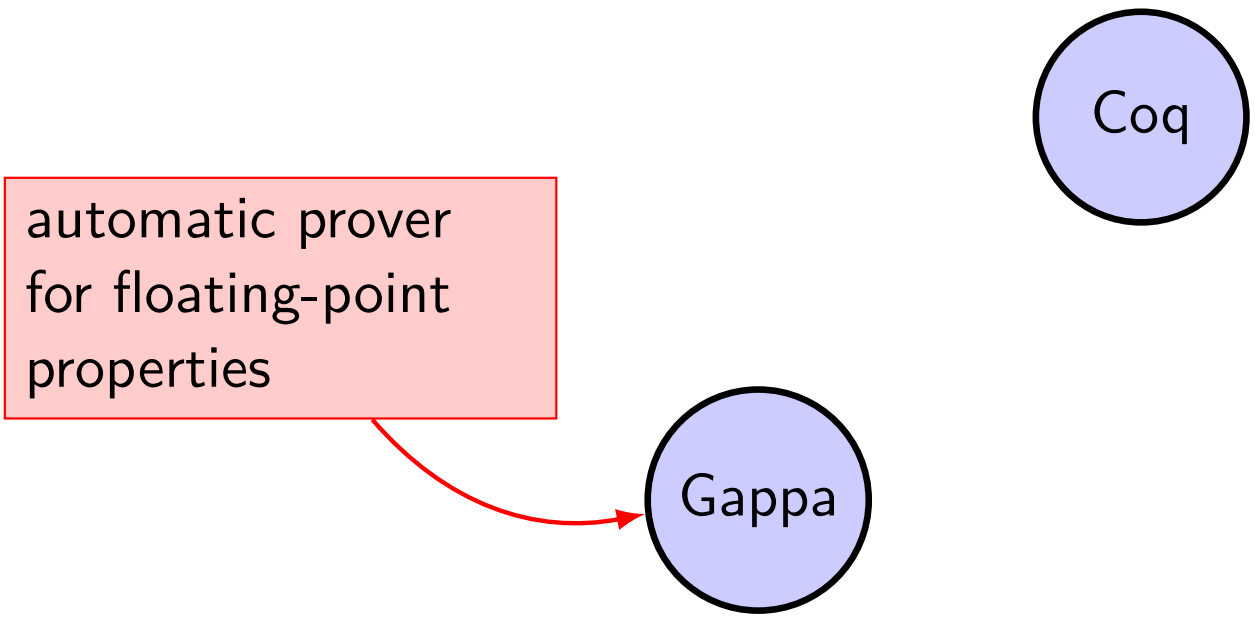
Can proofs that rely on numerical computations be redeemed?

Outline

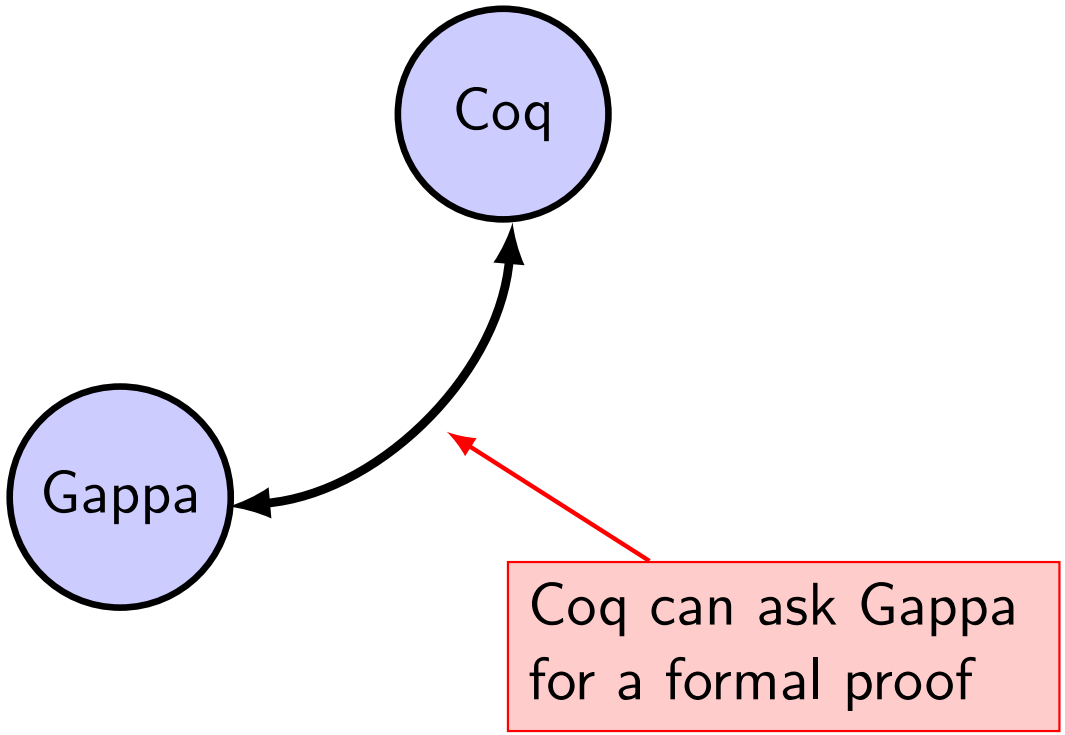
Some Relevant Tools



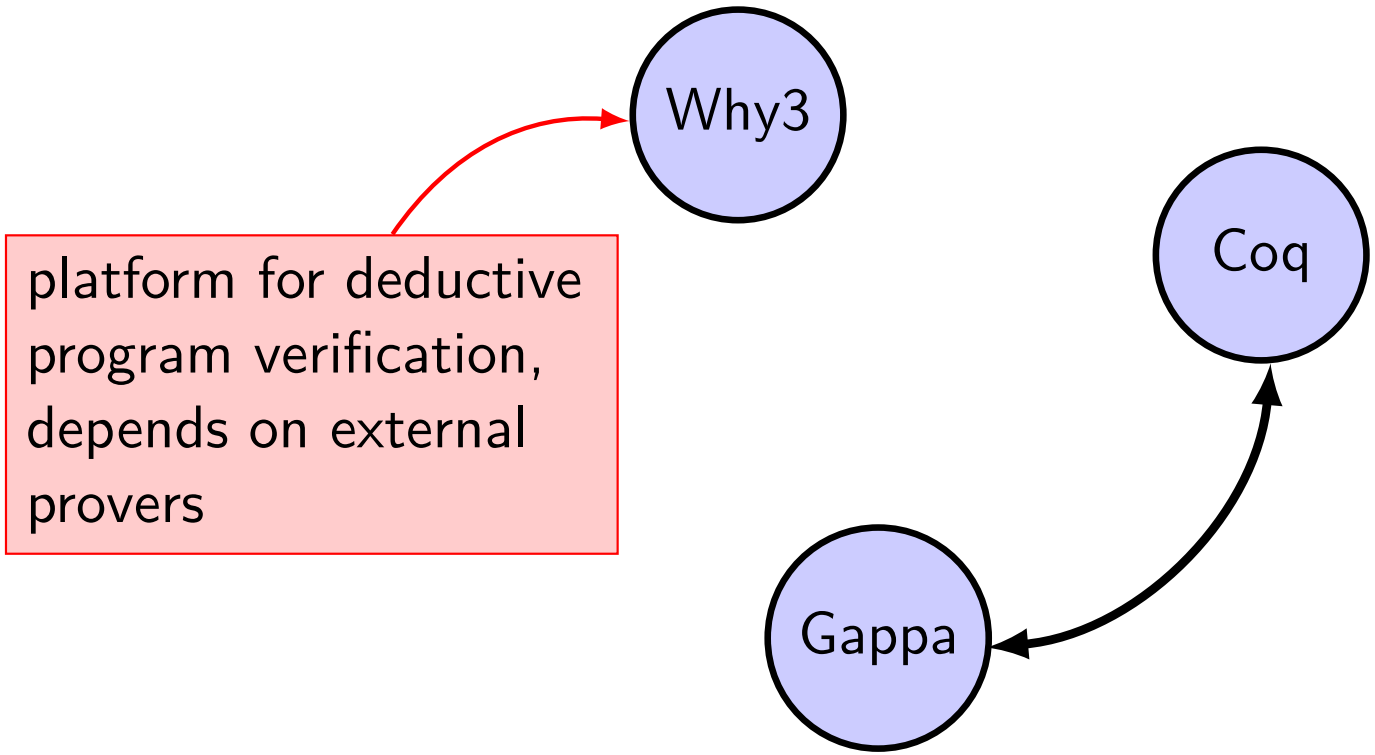
Some Relevant Tools



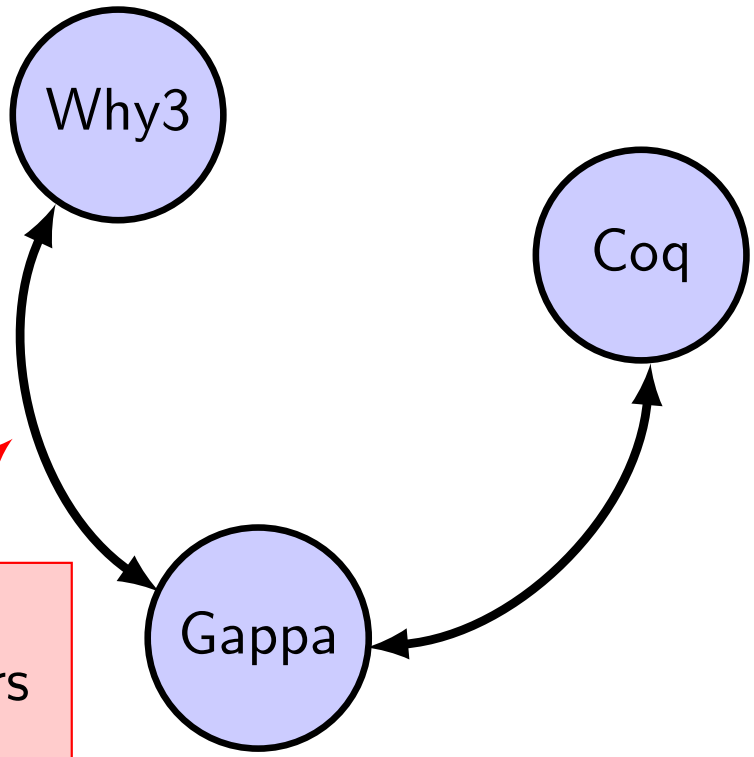
Some Relevant Tools



Some Relevant Tools

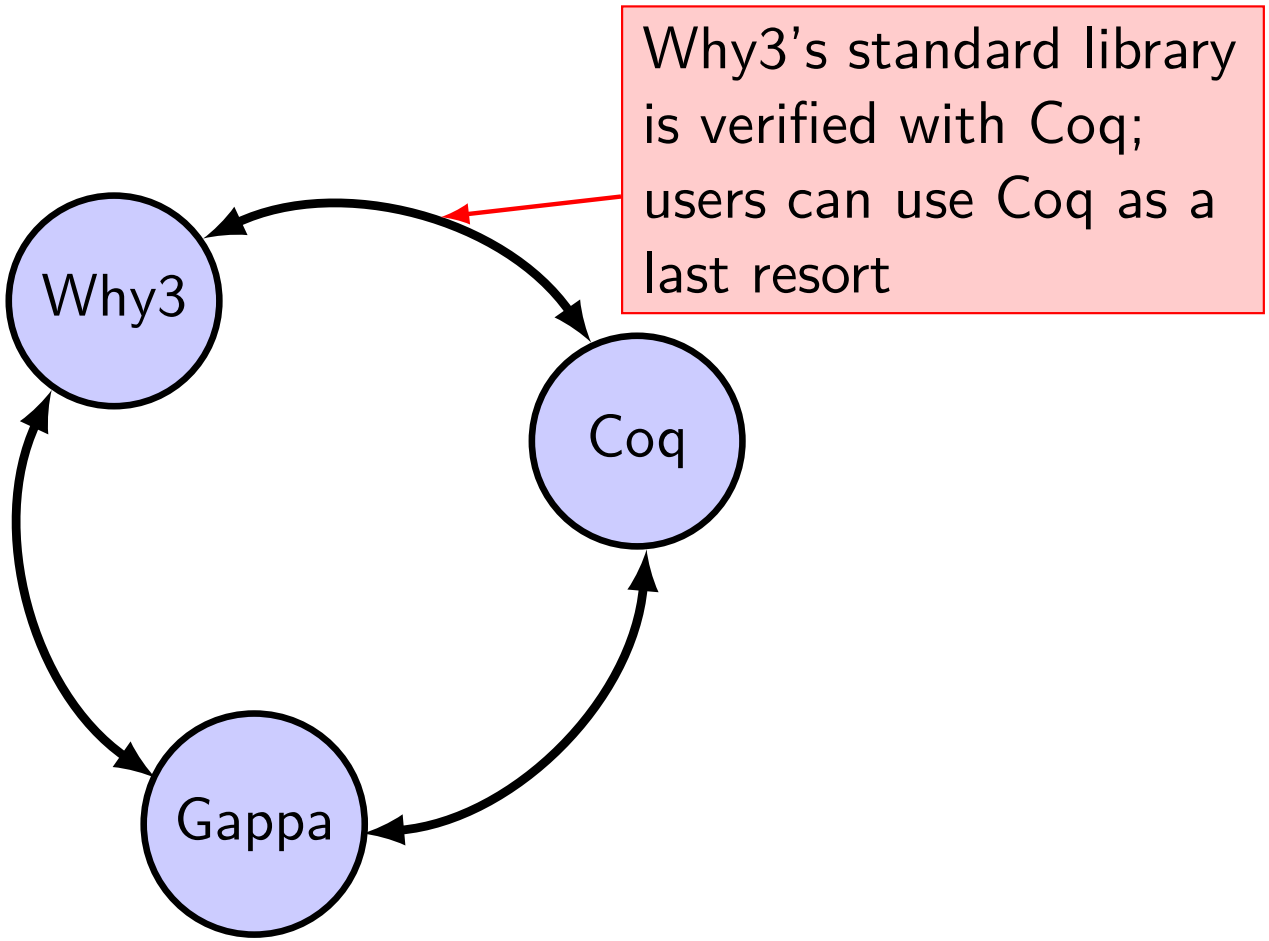


Some Relevant Tools



Gappa is one of the many automated solvers supported by Why3

Some Relevant Tools



Outline

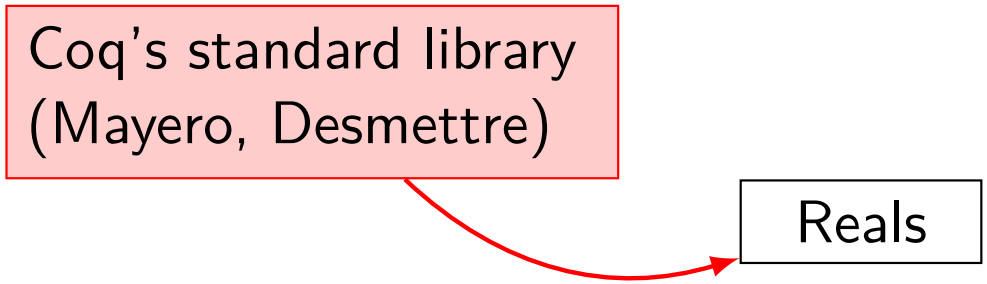
Numerical Computations for Formal Verification

Objective

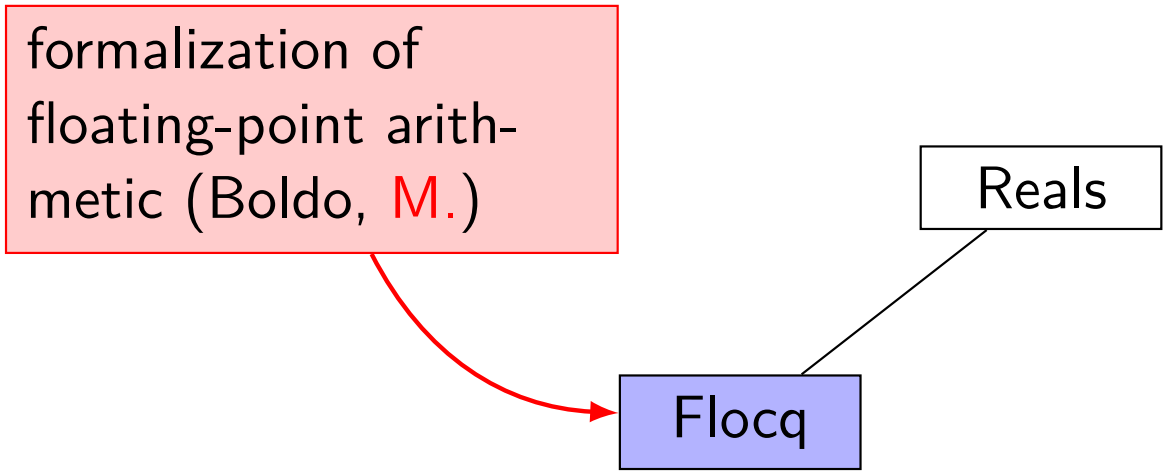
Can we produce a convincing proof of

$$\int_{-\infty}^{\infty} \frac{(0.5 \cdot \log(\tau^2 + 2.25) + 4.1396 + \log \pi)^2}{0.25 + \tau^2} d\tau \leq 226.844 \ \textbf{?}$$

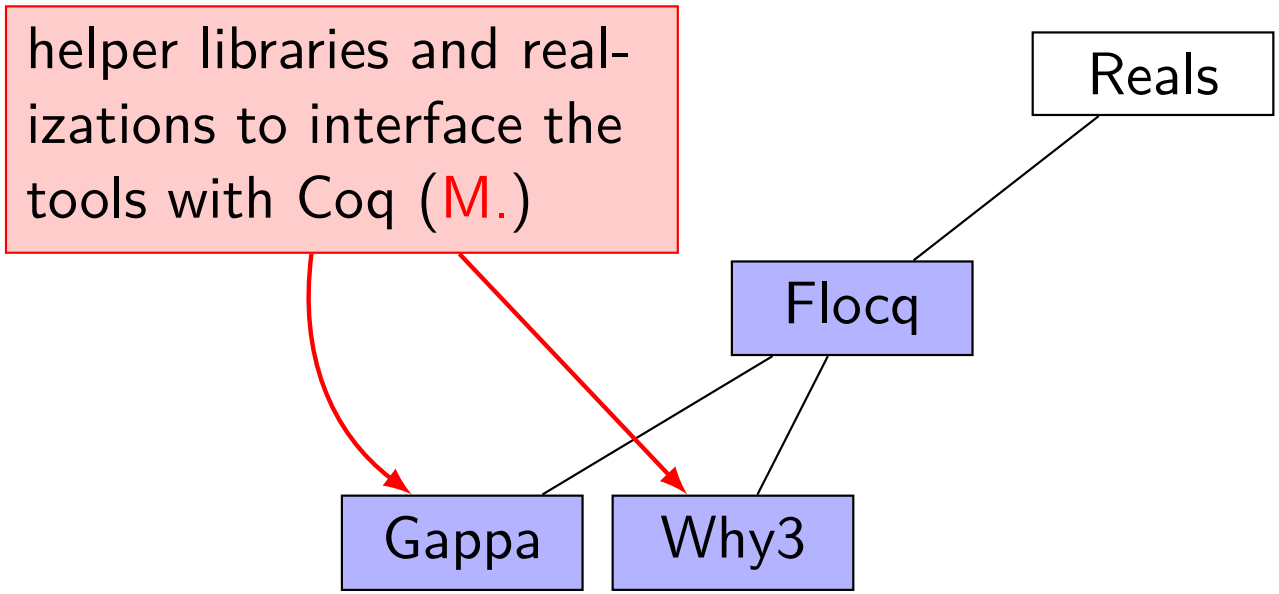
Some Relevant Coq Libraries



Some Relevant Coq Libraries

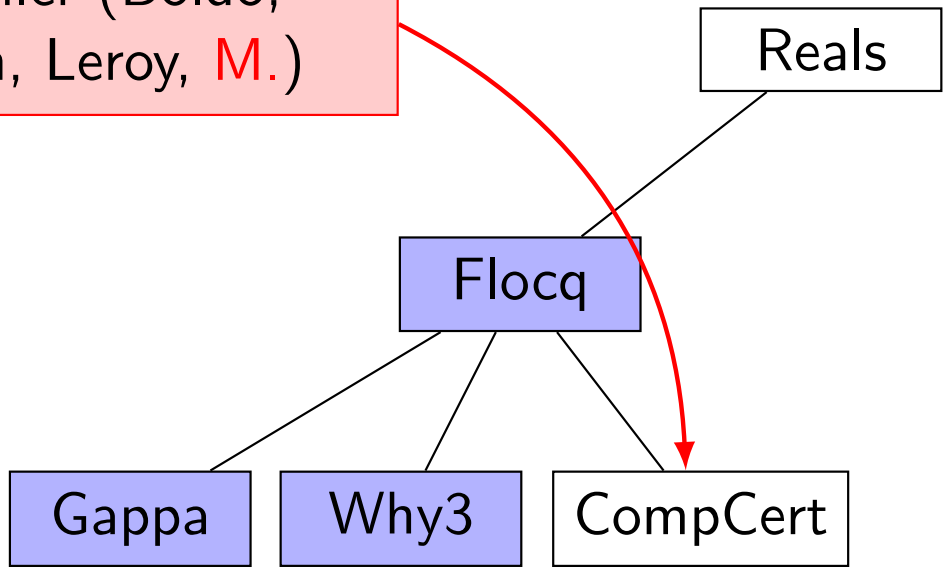


Some Relevant Coq Libraries

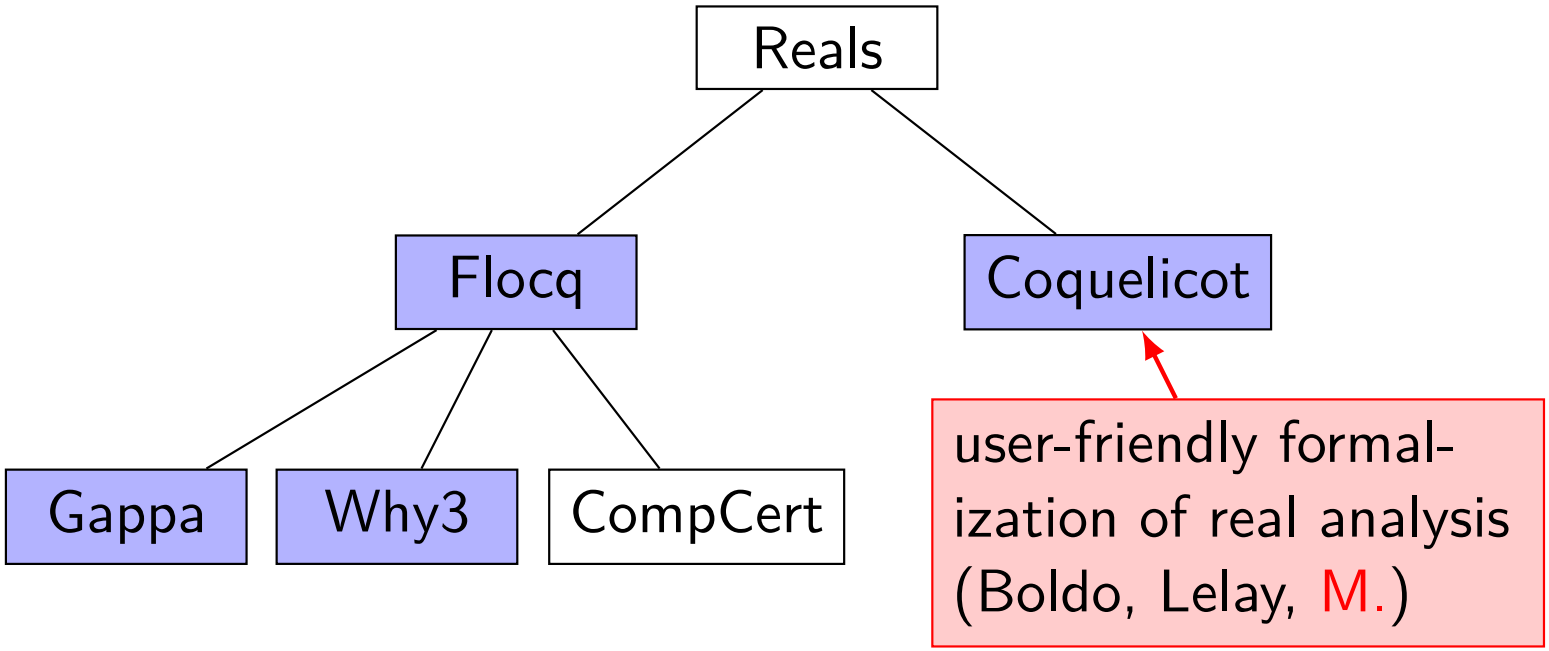


Some Relevant Coq Libraries

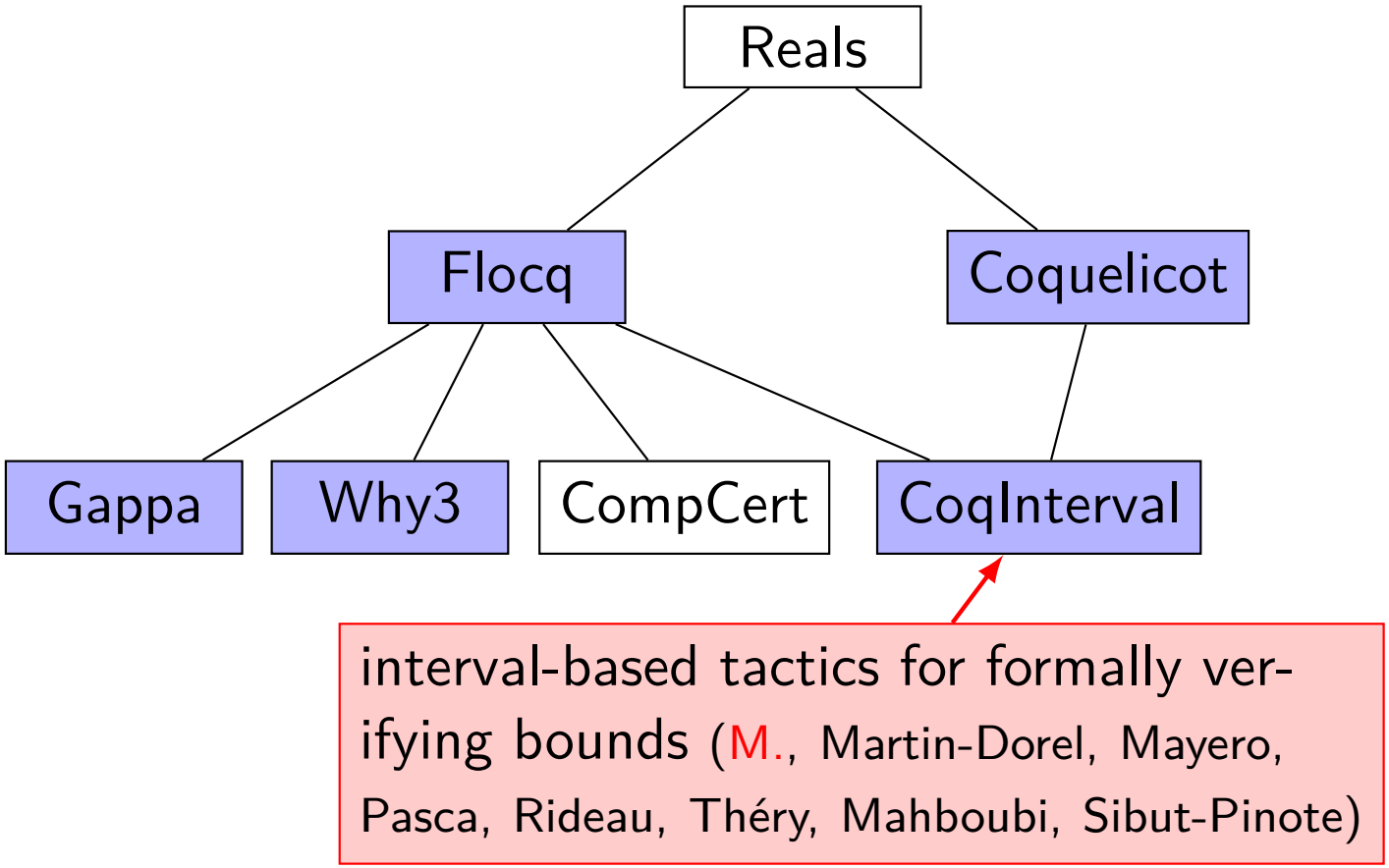
IEEE-754 semantics preservation for a C compiler (Boldo, Jourdan, Leroy, M.)



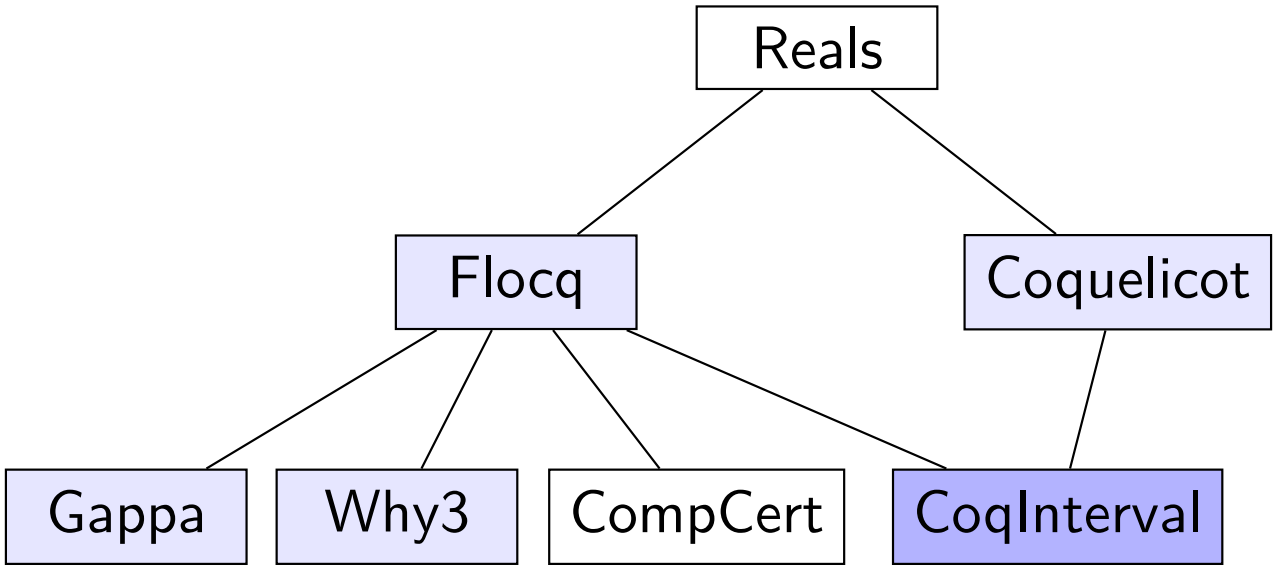
Some Relevant Coq Libraries



Some Relevant Coq Libraries



Some Relevant Coq Libraries



Example: a Proper Definite Integral

Example (Mathematics and Coq script)

$$\int_0^1 \tan t^2 \, dt \leq \frac{2}{5}.$$

Example: a Proper Definite Integral

Example (Mathematics and Coq script)

$$\int_0^1 \tan t^2 dt \leq \frac{2}{5}.$$

```
Goal RInt (fun t => tan (t*t)) 0 1 <= 2/5.
Proof. interval. Qed.
```

Example: a Proper Definite Integral

Example (Mathematics and Coq script)

integral from Coquelicot

$$\int_0^1 \tan t^2 dt \leq \frac{2}{5}.$$

```
Goal RInt (fun t => tan (t*t)) 0 1 <= 2/5.
Proof. interval. Qed.
```

tactic from CoqInterval

Formal proof is instant!

Interval Arithmetic

Definition (Intervals)

Connected closed subsets of $\mathbb{R}$, e.g., $\mathbf{x} = [\underline{x}; \overline{x}]$ or $[\underline{x}; +\infty)$.

Interval Arithmetic

Definition (Intervals)

Connected closed subsets of $\mathbb{R}$, e.g., $\mathbf{x} = [\underline{x}; \overline{x}]$ or $[\underline{x}; +\infty)$.

Definition (Interval extension)

$\mathbf{f} : \mathbb{I}^n \to \mathbb{I}$ is an interval extension of $f : \mathbb{R}^n \to \mathbb{R}$ if

$$\forall \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{I},\ \forall x_1, \ldots, x_n \in \mathbb{R},$$
$$x_1 \in \mathbf{x}_1 \wedge \ldots \wedge x_n \in \mathbf{x}_n \Rightarrow f(x_1, \ldots, x_n) \in \mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n).$$

Interval Arithmetic

Definition (Intervals)

Connected closed subsets of $\mathbb{R}$, e.g., $\mathbf{x} = [\underline{x}; \overline{x}]$ or $[\underline{x}; +\infty)$.

Definition (Interval extension)

$\mathbf{f} : \mathbb{I}^n \to \mathbb{I}$ is an interval extension of $f : \mathbb{R}^n \to \mathbb{R}$ if

$$\forall \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{I},\ \forall x_1, \ldots, x_n \in \mathbb{R},$$
$$x_1 \in \mathbf{x}_1 \wedge \ldots \wedge x_n \in \mathbf{x}_n \Rightarrow f(x_1, \ldots, x_n) \in \mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n).$$

Lemma (Naive quadrature)

$$\int_u^v f(t)\,dt \in (\mathbf{v} - \mathbf{u}) \cdot \mathbf{f}(\mathrm{hull}(\mathbf{u}, \mathbf{v})).$$

Effective Interval Extensions

Definition (Floating-point numbers and directed rounding)

$$\mathbb{F} = \{ m \cdot \beta^e \in \mathbb{R} \mid m, e \in \mathbb{Z} \wedge |m| < \beta^\varrho \}.$$

$$\forall u, v \in \mathbb{F}, \quad \bigtriangledown(u \diamond v) \leq u \diamond v \leq \bigtriangleup(u \diamond v).$$

Effective Interval Extensions

Definition (Floating-point numbers and directed rounding)

$$\mathbb{F} = \{m \cdot \beta^e \in \mathbb{R} \mid m, e \in \mathbb{Z} \wedge |m| < \beta^\varrho\}.$$

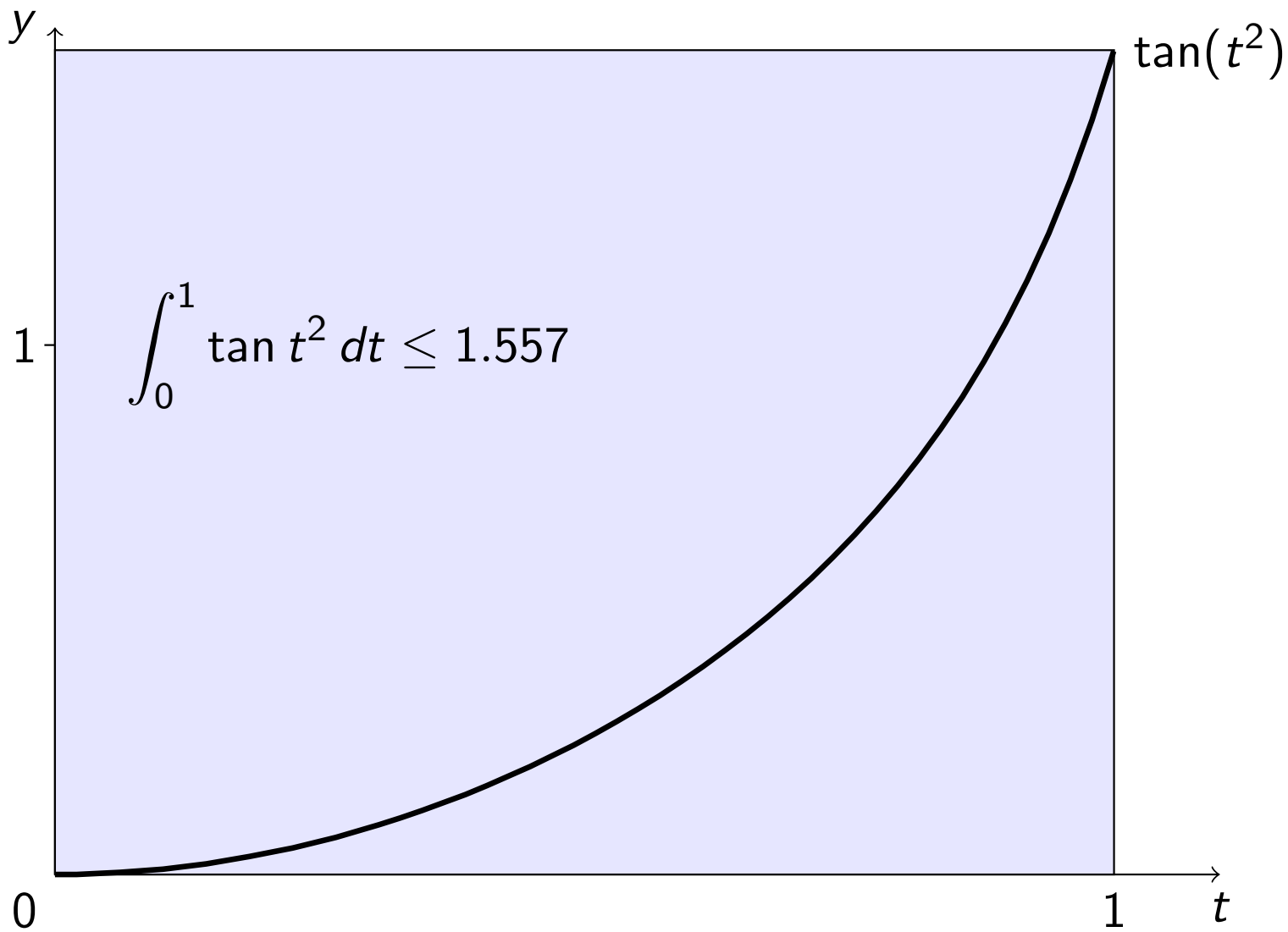
$$\forall u, v \in \mathbb{F}, \quad \bigtriangledown(u \diamond v) \leq u \diamond v \leq \bigtriangleup(u \diamond v).$$

Interval extensions of $+$, $-$, $\times$

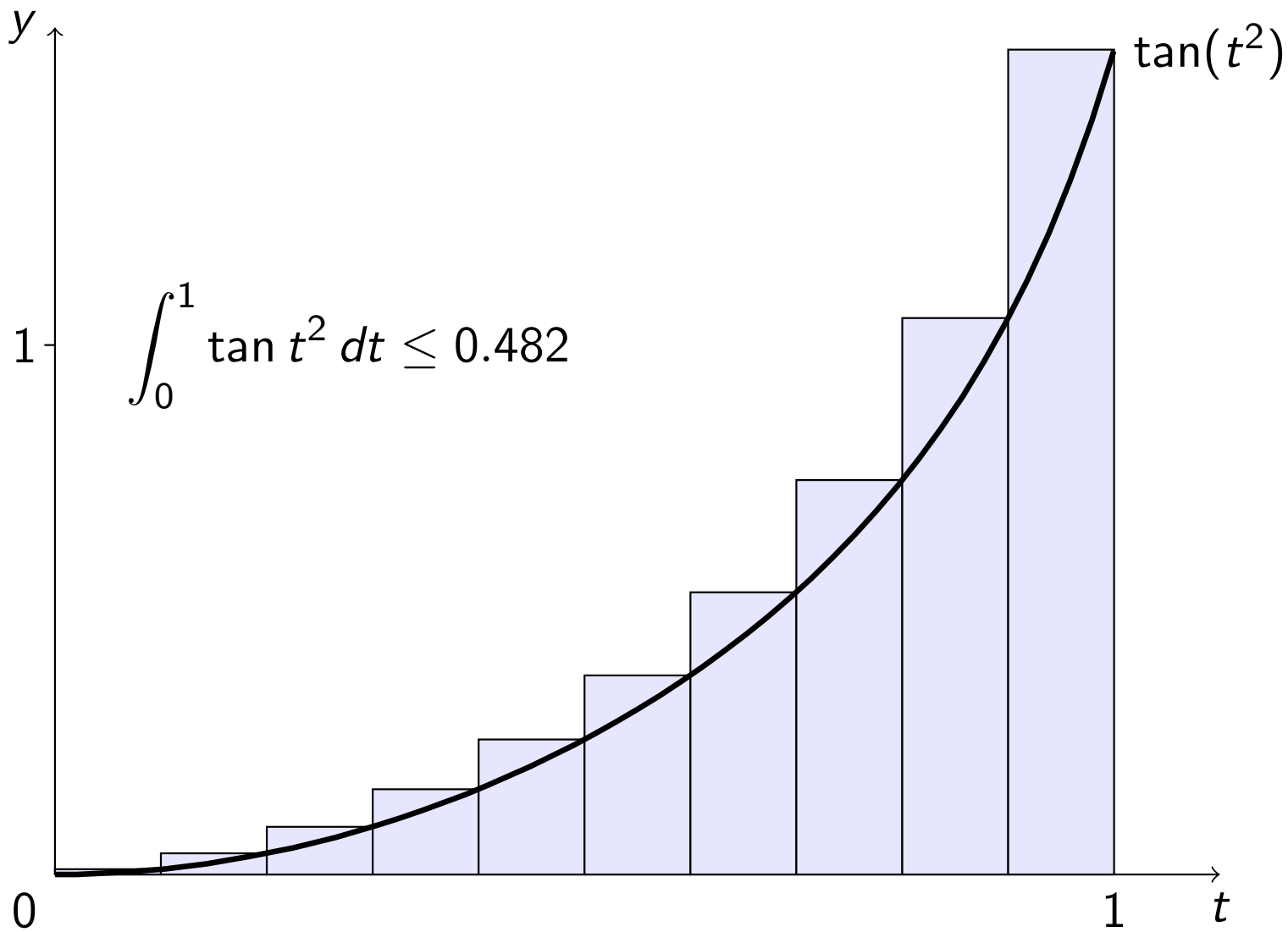
If $u \in \mathbf{u} = [\underline{u}; \overline{u}]$ and $v \in \mathbf{v} = [\underline{v}; \overline{v}]$, then (by monotony)

$$\begin{aligned}
u + v &\in [\bigtriangledown(\underline{u} + \underline{v}); \bigtriangleup(\overline{u} + \overline{v})] \stackrel{\text{def}}{=} \mathbf{u} + \mathbf{v}, \\
u - v &\in [\bigtriangledown(\underline{u} - \overline{v}); \bigtriangleup(\overline{u} - \underline{v})] \stackrel{\text{def}}{=} \mathbf{u} - \mathbf{v}, \\
u \cdot v &\in [\min(\bigtriangledown(\underline{u} \cdot \underline{v}), \bigtriangledown(\underline{u} \cdot \overline{v}), \bigtriangledown(\overline{u} \cdot \underline{v}), \bigtriangledown(\overline{u} \cdot \overline{v})); \\
&\qquad \max(\bigtriangleup(\underline{u} \cdot \underline{v}), \bigtriangleup(\underline{u} \cdot \overline{v}), \bigtriangleup(\overline{u} \cdot \underline{v}), \bigtriangleup(\overline{u} \cdot \overline{v}))].
\end{aligned}$$

Quadrature



Quadrature



Formally Reliable Computations

What is needed?

1. Formalize floating-point rounding
   and verify basic arithmetic operators.

Formally Reliable Computations

What is needed?

1. Formalize floating-point rounding
   and verify basic arithmetic operators.
2. Verify basic interval operators and automatic differentiation.

Formally Reliable Computations

What is needed?

1. Formalize floating-point rounding
   and verify basic arithmetic operators.
2. Verify basic interval operators and automatic differentiation.
3. Formalize real analysis: power series, integrals, etc.

Formally Reliable Computations

What is needed?

1. Formalize floating-point rounding
   and verify basic arithmetic operators.
2. Verify basic interval operators and automatic differentiation.
3. Formalize real analysis: power series, integrals, etc.
4. Verify interval extensions of elementary functions.

Formally Reliable Computations

What is needed?

1. Formalize floating-point rounding
   and verify basic arithmetic operators.
2. Verify basic interval operators and automatic differentiation.
3. Formalize real analysis: power series, integrals, etc.
4. Verify interval extensions of elementary functions.
5. Support rigorous polynomial approximations
   based on Taylor models.

Formally Reliable Computations

What is needed?

1. Formalize floating-point rounding
   and verify basic arithmetic operators.
2. Verify basic interval operators and automatic differentiation.
3. Formalize real analysis: power series, integrals, etc.
4. Verify interval extensions of elementary functions.
5. Support rigorous polynomial approximations
   based on Taylor models.
6. Verify quadrature algorithms
   for proper and improper integrals.

Formally Reliable Computations

What is needed?

1. Formalize floating-point rounding
   and verify basic arithmetic operators. [ARITH'11]
2. Verify basic interval operators and automatic differentiation. [IJCAR'08]
3. Formalize real analysis: power series, integrals, etc. [CPP'12, MCS'15]
4. Verify interval extensions of elementary functions. [RNC'08, I&C'12]
5. Support rigorous polynomial approximations
   based on Taylor models. [JAR'16]
6. Verify quadrature algorithms
   for proper and improper integrals. [ITP'16, JAR'18]

Details: Improving Enclosures

Definition (Polynomial enclosure)

$(\vec{\mathbf{p}}, \mathbf{\Delta}) \in \mathbb{I}^{n+1} \times \mathbb{I}$ encloses f over $\mathbf{x} \ni x_0$, if $\exists p \in \mathbb{R}[X]$,

$$(\forall i < n,\ p_i \in \mathbf{p}_i) \ \wedge\ \forall x \in \mathbf{x},\ f(x) - p(x - x_0) \in \mathbf{\Delta}.$$

Details: Improving Enclosures

Definition (Polynomial enclosure)

$(\vec{\mathbf{p}}, \mathbf{\Delta}) \in \mathbb{I}^{n+1} \times \mathbb{I}$ encloses f over $\mathbf{x} \ni x_0$, if $\exists p \in \mathbb{R}[X]$,

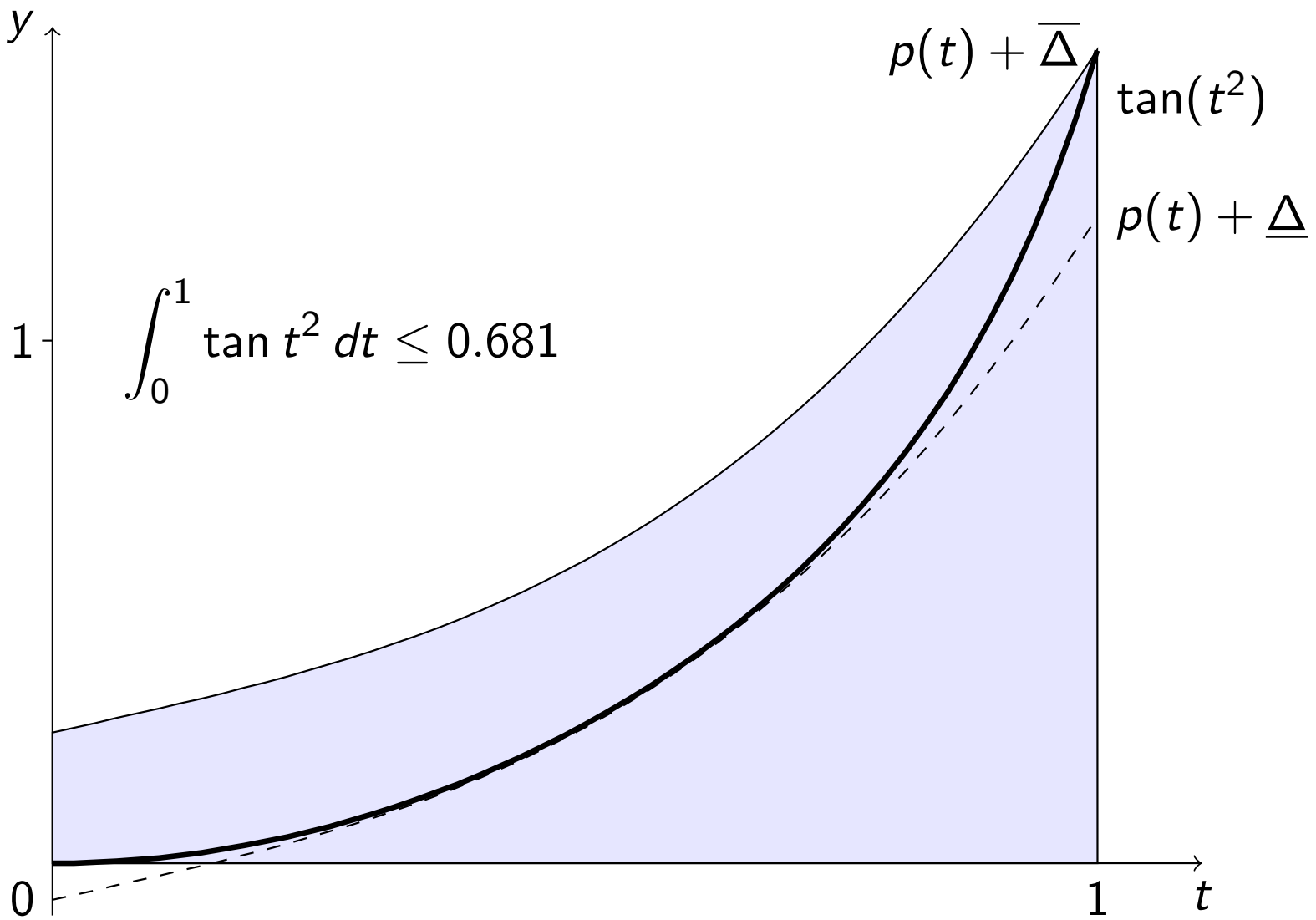
$$(\forall i < n,\ p_i \in \mathbf{p}_i) \ \wedge\ \forall x \in \mathbf{x},\ f(x) - p(x - x_0) \in \mathbf{\Delta}.$$

Lemma (Improved quadrature)

If $(p, \mathbf{\Delta})$ encloses f over $[u; v]$, and if P is a primitive of p, then

$$\int_u^v f \in P(v) - P(u) + (v - u) \cdot \mathbf{\Delta}.$$

Quadrature with a Degree-3 Polynomial



Helfgott's Integrals

Proper integral, in the MathOverflow post

$$\int_0^1 \left|\left(x^4 + 10x^3 + 19x^2 - 6x - 6\right)\, e^x\right| dx \simeq 11.14731055.$$

Helfgott's Integrals

Proper integral, in the MathOverflow post

$$\int_0^1 \left|\left(x^4 + 10x^3 + 19x^2 - 6x - 6\right)\, e^x\right| dx \simeq 11.14731055.$$

INTLAB got this integral wrong.
VNODE/LP cannot compute it.

Helfgott's Integrals

Proper integral, in the MathOverflow post

$$\int_0^1 \left|\left(x^4 + 10x^3 + 19x^2 - 6x - 6\right)\, e^x\right| dx \simeq 11.14731055.$$

INTLAB got this integral wrong.
VNODE/LP cannot compute it.

Improper integral, in the proof

$$\int_{-\infty}^{+\infty} \frac{(0.5 \cdot \log(\tau^2 + 2.25) + 4.1396 + \log \pi)^2}{0.25 + \tau^2}\, d\tau$$

$$\in [226.849; 226.850].$$

Helfgott's Integrals

Proper integral, in the MathOverflow post

$$\int_0^1 \left|\left(x^4 + 10x^3 + 19x^2 - 6x - 6\right)\, e^x\right| dx \simeq 11.14731055.$$

INTLAB got this integral wrong.
VNODE/LP cannot compute it.

Improper integral, in the proof

$$\int_{-\infty}^{+\infty} \frac{(0.5 \cdot \log(\tau^2 + 2.25) + 4.1396 + \log \pi)^2}{0.25 + \tau^2}\, d\tau$$

$$\in [226.849; 226.850].$$

Helfgott was using ≤ 226.844.

Outline

Arbitrary-Precision Integer Arithmetic

The GNU Multiple Precision arithmetic library (GMP)

- Free software, widely used.
- State-of-the-art algorithms, unmatched performances.

Arbitrary-Precision Integer Arithmetic

The GNU Multiple Precision arithmetic library (GMP)

- Free software, widely used.
- State-of-the-art algorithms, unmatched performances.

- Highly intricate algorithms written in low-level C and ASM.
- Ill-suited for random testing.

  GMP 5.0.4: "Two bugs in multiplication [...] with extremely low probability [...]. Two bugs in the gcd code [...] For uniformly distributed random operands, the likelihood is infinitesimally small."

Arbitrary-Precision Integer Arithmetic

The GNU Multiple Precision arithmetic library (GMP)

- Free software, widely used.
- State-of-the-art algorithms, unmatched performances.

- Highly intricate algorithms written in low-level C and ASM.
- Ill-suited for random testing.

  GMP 5.0.4: "Two bugs in multiplication [...] with extremely low probability [...]. Two bugs in the gcd code [...] For uniformly distributed random operands, the likelihood is infinitesimally small."

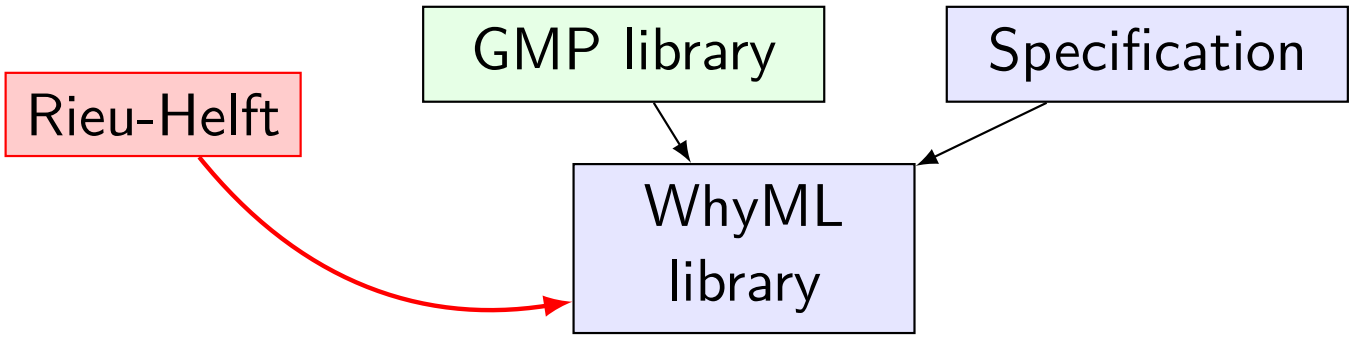
Objectives

- Produce a verified library compatible with GMP.
- Attain performances comparable to a no-assembly GMP.
- Focus on the low-level `mpn` layer.

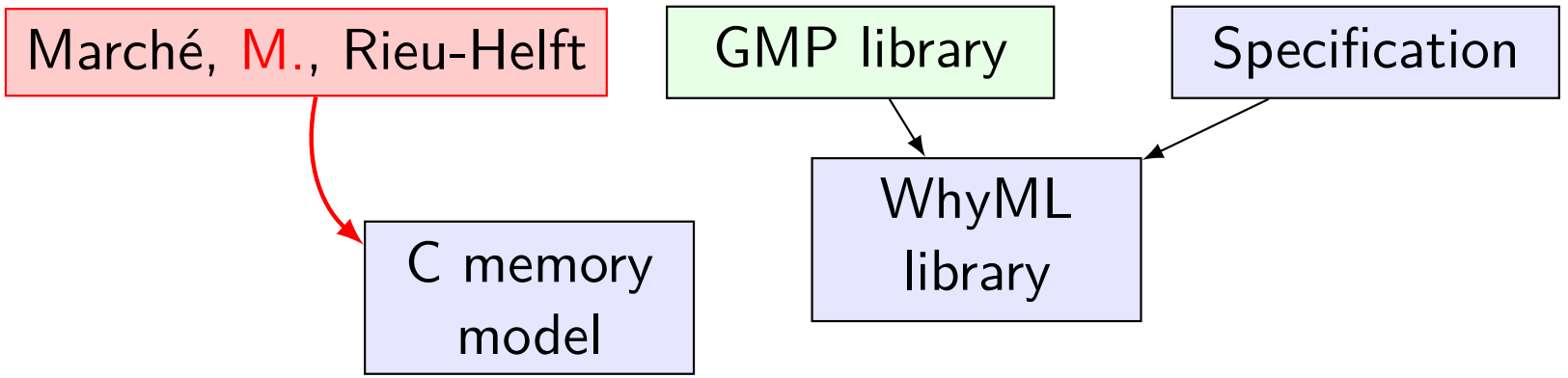
The Why3 Workflow

GMP library

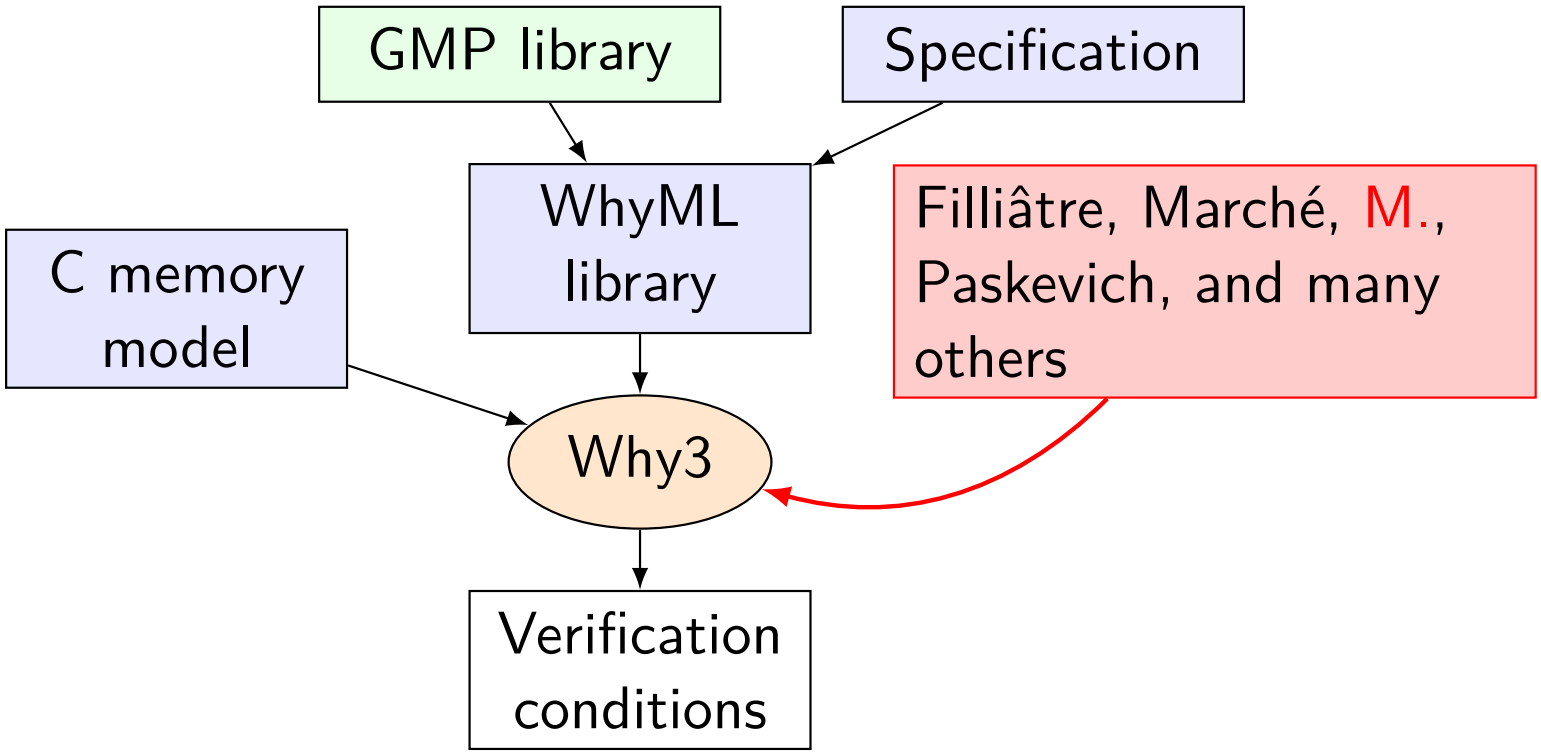
The Why3 Workflow



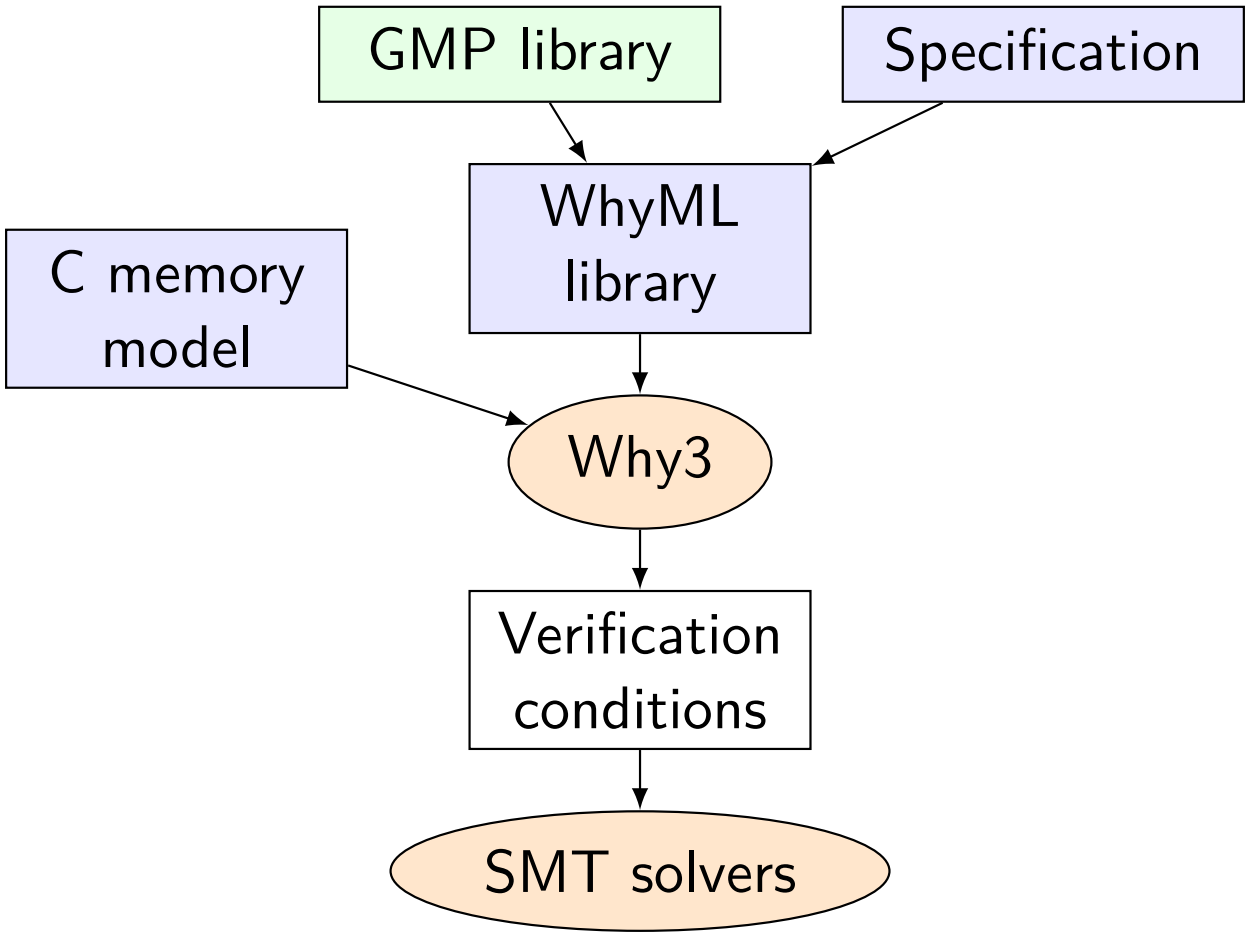
The Why3 Workflow



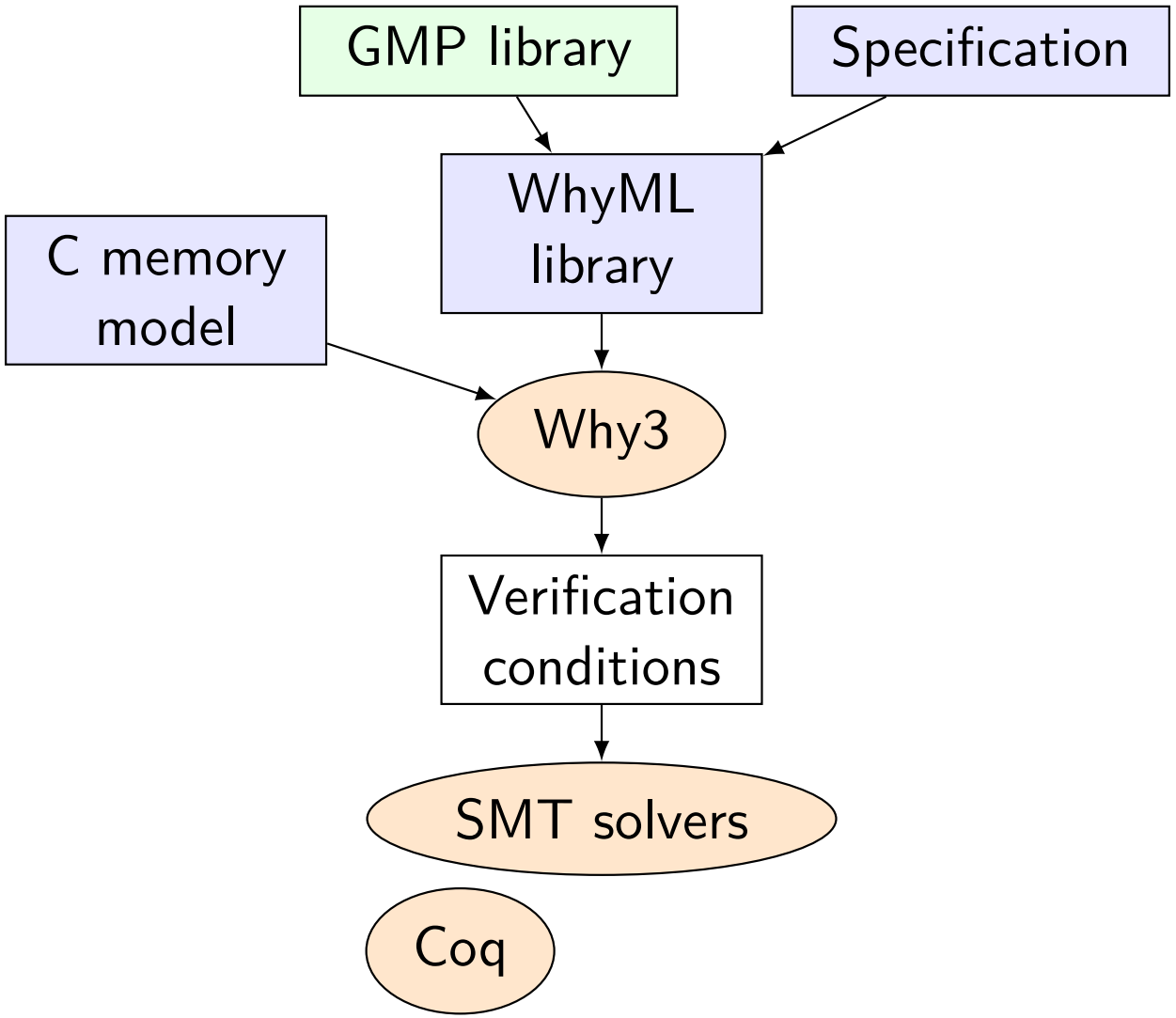
The Why3 Workflow



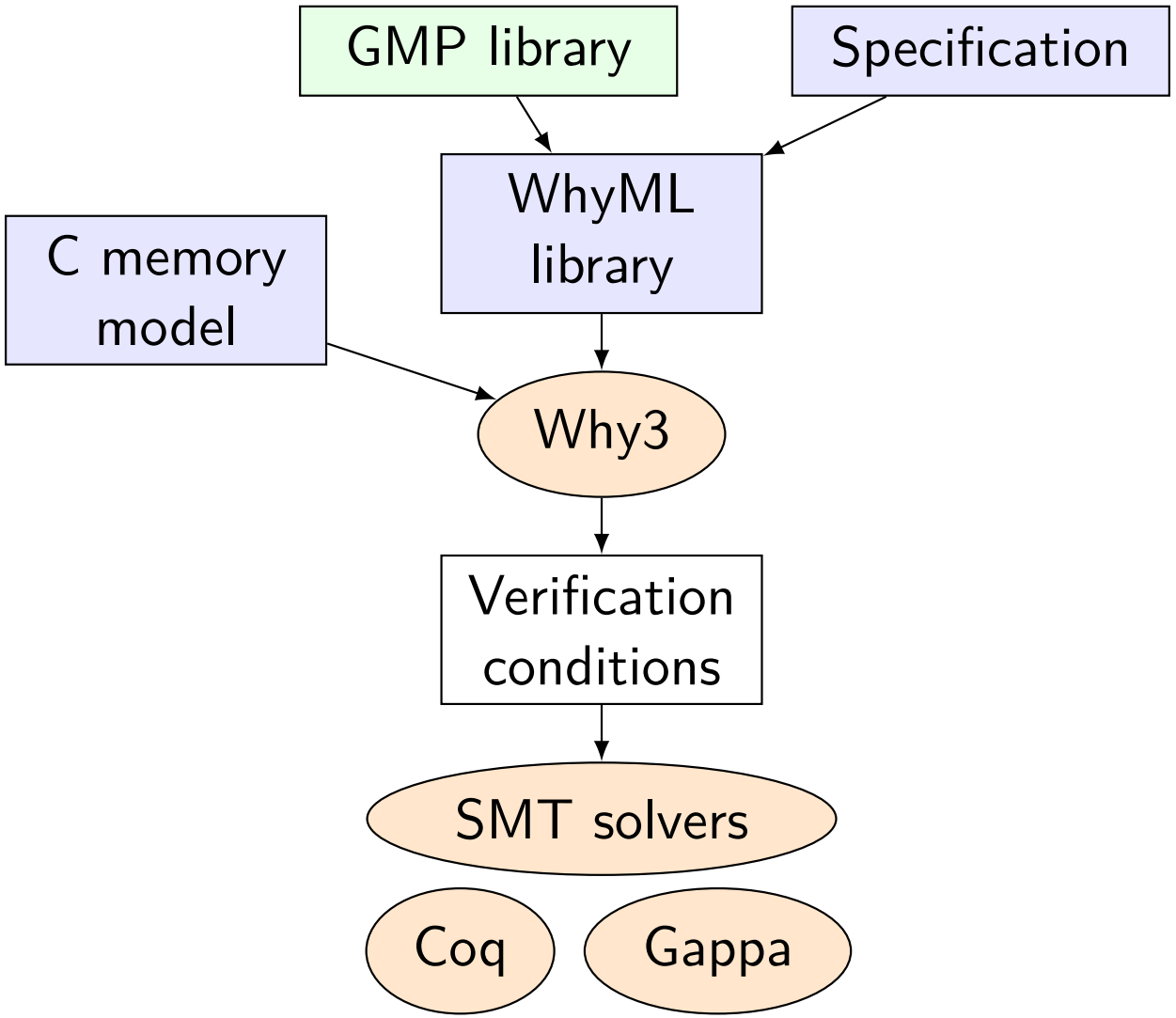
The Why3 Workflow



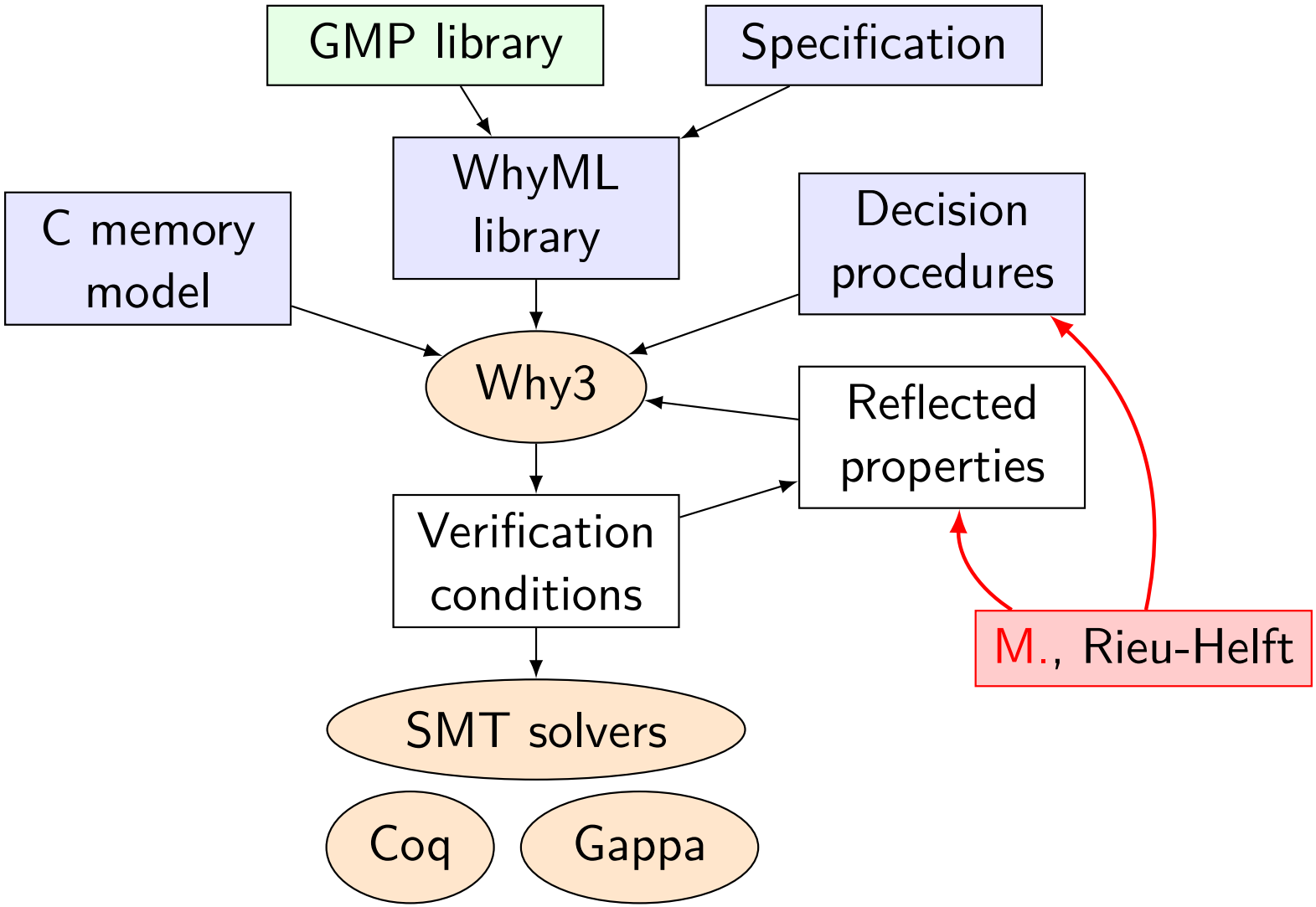
The Why3 Workflow



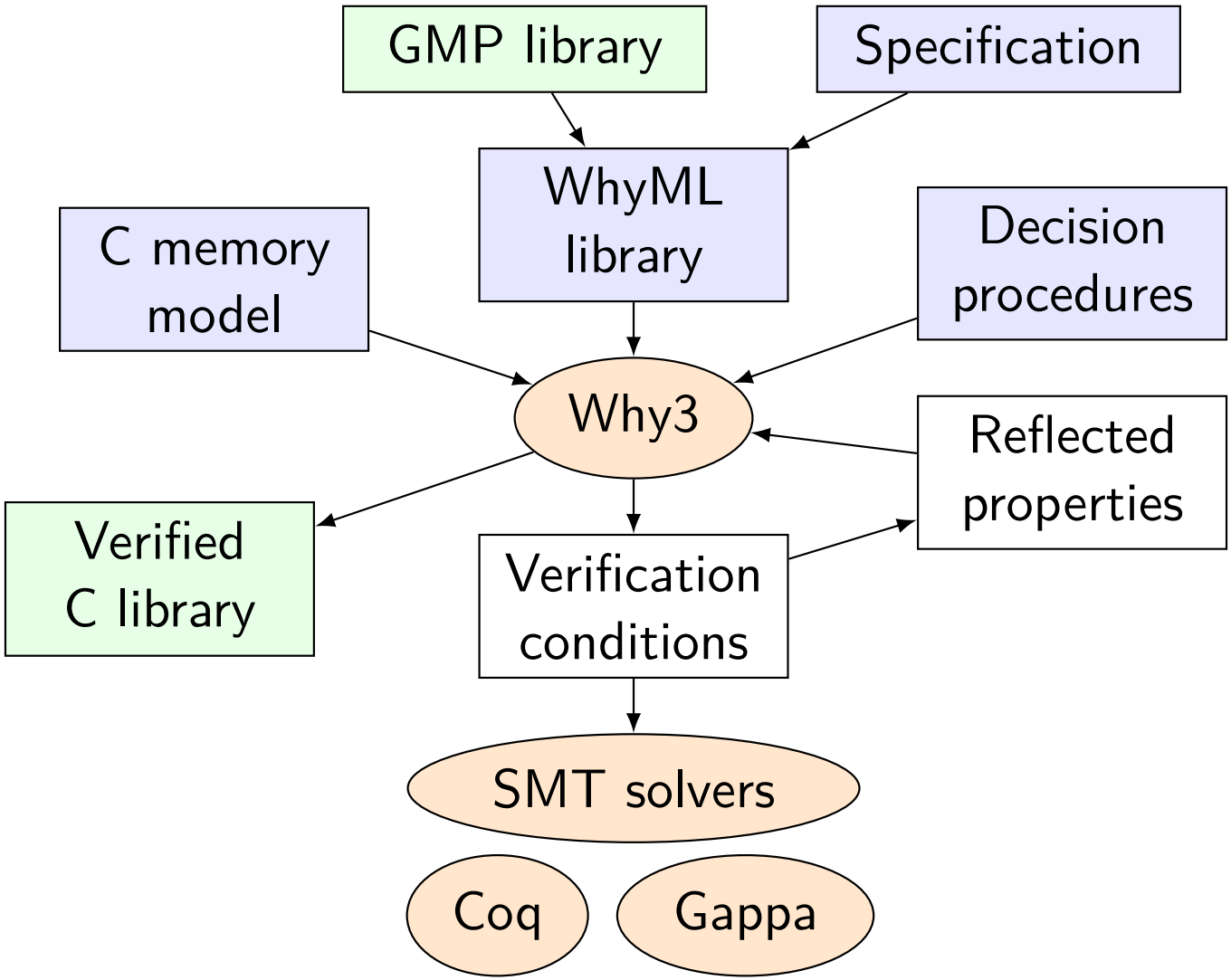
The Why3 Workflow



The Why3 Workflow



The Why3 Workflow



Example: Subtracting 1 to a Long Integer

Original macro (simplified from 18-line `mpn_decr_u`)

```
#define mpn_decr_1(p)                   \
  mp_ptr __p = (p);                     \
  while ((*(__p++))-- == 0) ;
```

Example: Subtracting 1 to a Long Integer

Original macro (simplified from 18-line `mpn_decr_u`)

```
#define mpn_decr_1(p)                   \
  mp_ptr __p = (p);                     \
  while ((*(__p++))-- == 0) ;
```

Conversion to WhyML

```
let wmpn_decr_1 (p: ptr uint64) : unit
=
  let ref lp = 0 in
  let ref i = 0 in
  while lp = 0 do
    lp <- get_ofs p i;
    set_ofs p i (sub_mod lp 1);
    i <- i + 1;
  done
```

Example: Subtracting 1 to a Long Integer

Original macro (simplified from 18-line `mpn_decr_u`)

```
#define mpn_decr_1(p)                   \
  mp_ptr __p = (p);                     \
  while ((*(__p++))-- == 0) ;
```

Conversion to WhyML

```
let wmpn_decr_1 (p: ptr uint64) : unit
  ensures  { value p = value (old p) - 1 }
=
  let ref lp = 0 in
  let ref i = 0 in
  while lp = 0 do
    lp <- get_ofs p i;
    set_ofs p i (sub_mod lp 1);
    i <- i + 1;
  done
```

Example: Subtracting 1 to a Long Integer

Original macro (simplified from 18-line `mpn_decr_u`)

```
#define mpn_decr_1(p)                   \
  mp_ptr __p = (p);                     \
  while ((*(__p++))-- == 0) ;
```

Conversion to WhyML

```
let wmpn_decr_1 (p: ptr uint64) (ghost sz: int32) : unit
  ensures  { value p sz = value (old p) sz - 1 }
=
  let ref lp = 0 in
  let ref i = 0 in
  while lp = 0 do
    lp <- get_ofs p i;
    set_ofs p i (sub_mod lp 1);
    i <- i + 1;
  done
```

Example: Subtracting 1 to a Long Integer

Original macro (simplified from 18-line `mpn_decr_u`)

```
#define mpn_decr_1(p)                   \
  mp_ptr __p = (p);                     \
  while ((*(__p++))-- == 0) ;
```

Conversion to WhyML

```
let wmpn_decr_1 (p: ptr uint64) (ghost sz: int32) : unit
  requires { valid p sz }
  requires { 1 <= value p sz }
  ensures  { value p sz = value (old p) sz - 1 }
=
  let ref lp = 0 in
  let ref i = 0 in
  while lp = 0 do
  ...
```

Example: Subtracting 1 to a Long Integer

Original macro (simplified from 18-line `mpn_decr_u`)

```
#define mpn_decr_1(p)                   \
  mp_ptr __p = (p);                     \
  while ((*(__p++))-- == 0) ;
```

Extraction to C

```
void wmpn_decr_1(uint64_t * p) {
  uint64_t lp; int32_t i;
  lp = 0;
  i = 0;
  while (lp == 0) {
    lp = p[i];
    p[i] = lp - 1;
    i = i + 1;
  }
}
```

Example: 64-bit Square Root

Original GMP code

```
mp_limb_t mpn_sqrtrem1(mp_ptr rp, mp_limb_t a0) {
  mp_limb_t a1, x0, t2, t, x2;
  unsigned abits = a0 >> (GMP_LIMB_BITS - 1 - 8);
  x0 = 0x100 | invsqrttab[abits - 0x80];
  /* x0 is now an 8 bits approximation of 1/sqrt(a0) */
  a1 = a0 >> (GMP_LIMB_BITS - 1 - 32);
  t = (mp_limb_signed_t) (CNST_LIMB(0x2000000000000)
        - 0x30000 - a1 * x0 * x0) >> 16;
  x0 = (x0<<16) + ((mp_limb_signed_t)(x0*t) >> (16+2));
  /* x0 is now a 16 bits approximation of 1/sqrt(a0) */
  ...
```

Example: 64-bit Square Root

Original GMP code

```
mp_limb_t mpn_sqrtrem1(mp_ptr rp, mp_limb_t a0) {
  mp_limb_t a1, x0, t2, t, x2;
  unsigned abits = a0 >> (GMP_LIMB_BITS - 1 - 8);
  x0 = 0x100 | invsqrttab[abits - 0x80];
  /* x0 is now an 8 bits approximation of 1/sqrt(a0) */
  a1 = a0 >> (GMP_LIMB_BITS - 1 - 32);
  t = (mp_limb_signed_t) (CNST_LIMB(0x2000000000000)
        - 0x30000 - a1 * x0 * x0) >> 16;
  x0 = (x0<<16) + ((mp_limb_signed_t)(x0*t) >> (16+2));
  /* x0 is now a 16 bits approximation of 1/sqrt(a0) */
  ...
```

- Table lookup, Newton iteration toward $1/\sqrt{a}$,
  modified Newton iteration toward $a/\sqrt{a}$, correcting step.

Example: 64-bit Square Root

Original GMP code

```
mp_limb_t mpn_sqrtrem1(mp_ptr rp, mp_limb_t a0) {
  mp_limb_t a1, x0, t2, t, x2;
  unsigned abits = a0 >> (GMP_LIMB_BITS - 1 - 8);
  x0 = 0x100 | invsqrttab[abits - 0x80];
  /* x0 is now an 8 bits approximation of 1/sqrt(a0) */
  a1 = a0 >> (GMP_LIMB_BITS - 1 - 32);
  t = (mp_limb_signed_t) (CNST_LIMB(0x2000000000000)
        - 0x30000 - a1 * x0 * x0) >> 16;
  x0 = (x0<<16) + ((mp_limb_signed_t)(x0*t) >> (16+2));
  /* x0 is now a 16 bits approximation of 1/sqrt(a0) */
  ...
```

- Table lookup, Newton iteration toward $1/\sqrt{a}$,
  modified Newton iteration toward $a/\sqrt{a}$, correcting step.
- Hand-coded fixed-point arithmetic.

A Small Fixed-Point Arithmetic Theory for Why3

```
type fxp = { ival: uint64;
             ghost rval: real; ghost iexp: int }
  invariant { rval = floor_at rval iexp }
  invariant { ival = mod (floor (rval *. pow2 (-iexp)))
                         (uint64'maxInt + 1) }
```

A Small Fixed-Point Arithmetic Theory for Why3

```
type fxp = { ival: uint64;
             ghost rval: real; ghost iexp: int }
  invariant { rval = floor_at rval iexp }
  invariant { ival = mod (floor (rval *. pow2 (-iexp)))
                         (uint64'maxInt + 1) }
```

Converted WhyML code

```
let sqrt1 (rp: ptr uint64) (a0: uint64): uint64 =
  let a = fxp_init a0 (-64) in
  let x0 = rsa_estimate a in
  let a1 = fxp_lsr a 31 in
  let m1 = fxp_sub (fxp_init 0x2000000000000 (-49))
                   (fxp_init 0x30000 (-49)) in
  let t1' = fxp_sub m1 (fxp_mul (fxp_mul x0 x0) a1) in
  let t1 = fxp_asr t1' 16 in
  let x1 = fxp_add (fxp_lsl x0 16)
     (fxp_asr' (fxp_mul x0 t1) 18 1) in
  ...
```

Error Analysis

Newton iteration toward $1/\sqrt{a}$

- Recurrence: $x_{i+1} = x_i + x_i \cdot (1 - a \cdot x_i^2)/2$.
- Relative error: $x_i = a^{-1/2} \cdot (1 + \varepsilon_i)$.
- Quadratic convergence: $|\varepsilon_{i+1}| \leq \frac{3}{2}|\varepsilon_i|^2$.

Error Analysis

Newton iteration toward $1/\sqrt{a}$

- Recurrence: $x_{i+1} = x_i + x_i \cdot (1 - a \cdot x_i^2)/2$.
- Relative error: $x_i = a^{-1/2} \cdot (1 + \varepsilon_i)$.
- Quadratic convergence: $|\varepsilon_{i+1}| \leq \frac{3}{2}|\varepsilon_i|^2$.

But what the code actually computes is

$$\tilde{x}_1 = \tilde{x}_0 + \bigtriangledown_{-24}\left(\tilde{x}_0 \cdot \bigtriangledown_{-33}\left(1 - 3 \cdot 2^{-33} - \bigtriangledown_{-33}(a) \cdot \tilde{x}_0^2\right)/2\right),$$

with $\bigtriangledown_k(r) = \lfloor r \cdot 2^{-k} \rfloor \cdot 2^k$.

Error Analysis

Newton iteration toward $1/\sqrt{a}$

- Recurrence: $x_{i+1} = x_i + x_i \cdot (1 - a \cdot x_i^2)/2$.
- Relative error: $x_i = a^{-1/2} \cdot (1 + \varepsilon_i)$.
- Quadratic convergence: $|\varepsilon_{i+1}| \leq \frac{3}{2}|\varepsilon_i|^2$.

But what the code actually computes is

$$\tilde{x}_1 = \tilde{x}_0 + \nabla_{-24}\left(\tilde{x}_0 \cdot \nabla_{-33}\left(1 - 3 \cdot 2^{-33} - \nabla_{-33}(a) \cdot \tilde{x}_0^2\right)/2\right),$$

with $\nabla_k(r) = \lfloor r \cdot 2^{-k} \rfloor \cdot 2^k$.

What to do about...

- Rounding errors? That is what Gappa is designed to handle.

Error Analysis

Newton iteration toward $1/\sqrt{a}$

- Recurrence: $x_{i+1} = x_i + x_i \cdot (1 - a \cdot x_i^2)/2$.
- Relative error: $x_i = a^{-1/2} \cdot (1 + \varepsilon_i)$.
- Quadratic convergence: $|\varepsilon_{i+1}| \leq \frac{3}{2}|\varepsilon_i|^2$.

But what the code actually computes is

$$\tilde{x}_1 = \tilde{x}_0 + \nabla_{-24}\left(\tilde{x}_0 \cdot \nabla_{-33}\left(1 - 3 \cdot 2^{-33} - \nabla_{-33}(a) \cdot \tilde{x}_0^2\right)/2\right),$$

with $\nabla_k(r) = \lfloor r \cdot 2^{-k} \rfloor \cdot 2^k$.

What to do about...

- Rounding errors? That is what Gappa is designed to handle.
- Magical constants? Critical for soundness; hinted to Gappa.

Gappa, in a Nutshell

Proof search by saturating over ~ 200 theorems

- Interval arithmetic: $\forall u, v \in \mathbb{R},\ \forall \mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{I},$
  $u \in \mathbf{u} \wedge v \in \mathbf{v} \wedge \mathbf{u} + \mathbf{v} \subseteq \mathbf{w} \Rightarrow u + v \in \mathbf{w};$

Gappa, in a Nutshell

Proof search by saturating over ~ 200 theorems

- Interval arithmetic: $\forall u, v \in \mathbb{R},\ \forall \mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{I},$
  $u \in \mathbf{u} \wedge v \in \mathbf{v} \wedge \mathbf{u} + \mathbf{v} \subseteq \mathbf{w} \Rightarrow u + v \in \mathbf{w}$;
- Rounding properties: $\forall x \in \mathbb{R},\ \forall k \in \mathbb{Z},\ \forall \mathbf{e} \in \mathbb{I},$
  $[-2^k; 0] \subseteq \mathbf{e} \Rightarrow \bigtriangledown_k(x) - x \in \mathbf{e}$;

Gappa, in a Nutshell

Proof search by saturating over ~ 200 theorems

- Interval arithmetic: $\forall u, v \in \mathbb{R},\ \forall \mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{I},$
  $u \in \mathbf{u} \wedge v \in \mathbf{v} \wedge \mathbf{u} + \mathbf{v} \subseteq \mathbf{w} \Rightarrow u + v \in \mathbf{w}$;
- Rounding properties: $\forall x \in \mathbb{R},\ \forall k \in \mathbb{Z},\ \forall \mathbf{e} \in \mathbb{I},$
  $[-2^k; 0] \subseteq \mathbf{e} \Rightarrow \bigtriangledown_k(x) - x \in \mathbf{e}$;
- Error analysis: $\forall u, v, \tilde{u}, \tilde{v} \in \mathbb{R},$
  $\tilde{u} \cdot \tilde{v} - u \cdot v = (\tilde{u} - u) \cdot v + u \cdot (\tilde{v} - v) + (\tilde{u} - u) \cdot (\tilde{v} - v)$.

What if Gappa Fails?

Help it by providing equalities

```
let sqrt1 (rp: ptr uint64) (a0: uint64): uint64
  ensures {result*result <= a0 < (result+1)*(result+1)}
=
  ...
  let ghost rsa = pure { 1. /. sqrt a } in
  let ghost e0  = pure { (x0 -. rsa) /. rsa } in
  let ghost ea1 = pure { (a1 -. a) /. a } in
  let ghost mx1 = pure { x0 +. x0 *. t1' *. 0.5 } in
  assert { (mx1 -. rsa) /. rsa =
    -0.5 *. (e0*.e0 *. (3.+.e0) +. (1.+.e0) *.
      (1. -. m1 +. (1.+.e0)*.(1.+.e0) *. ea1)) };
  ...
```

What if Gappa Fails?

Help it by providing equalities

```
let sqrt1 (rp: ptr uint64) (a0: uint64): uint64
  ensures {result*result <= a0 < (result+1)*(result+1)}
=
  ...
  let ghost rsa = pure { 1. /. sqrt a } in
  let ghost e0  = pure { (x0 -. rsa) /. rsa } in
  let ghost ea1 = pure { (a1 -. a) /. a } in
  let ghost mx1 = pure { x0 +. x0 *. t1' *. 0.5 } in
  assert { (mx1 -. rsa) /. rsa =
    -0.5 *. (e0*.e0 *. (3.+.e0) +. (1.+.e0) *.
      (1. -. m1 +. (1.+.e0)*.(1.+.e0) *. ea1)) };
  ...
```

Four equalities are needed by Gappa:

- they hardly mention rounding errors;
- all of them are proved with straightforward Coq scripts.

A Verified Library

Supported operations

- Addition, subtraction, comparison.
- Multiplication: quadratic, and Toom-Cook 2 and 2.5.
- Division: “schoolbook”.
- Square root: divide-and-conquer.

Outline

Contributions

Floating-point arithmetic

- Formalization of high-level properties:
  rounding to odd, FMA, successor, etc.

Contributions

Floating-point arithmetic

- Formalization of high-level properties:
  rounding to odd, FMA, successor, etc.
- Formalization of IEEE-754 standard,
  used to implement a C compiler and verify its correctness.

Contributions

Floating-point arithmetic

- Formalization of high-level properties:
  rounding to odd, FMA, successor, etc.
- Formalization of IEEE-754 standard,
  used to implement a C compiler and verify its correctness.
- A procedure for Alt-Ergo and a theory for Why3.

Contributions

Floating-point arithmetic

- Formalization of high-level properties:
  rounding to odd, FMA, successor, etc.
- Formalization of IEEE-754 standard,
  used to implement a C compiler and verify its correctness.
- A procedure for Alt-Ergo and a theory for Why3.

Miscellaneous

- Formalization of real analysis, with a focus on usability.

Contributions

Floating-point arithmetic

- Formalization of high-level properties:
  rounding to odd, FMA, successor, etc.
- Formalization of IEEE-754 standard,
  used to implement a C compiler and verify its correctness.
- A procedure for Alt-Ergo and a theory for Why3.

Miscellaneous

- Formalization of real analysis, with a focus on usability.
- Verification of a 3-point scheme for the wave equation.

Contributions

Floating-point arithmetic

- Formalization of high-level properties:
  rounding to odd, FMA, successor, etc.
- Formalization of IEEE-754 standard,
  used to implement a C compiler and verify its correctness.
- A procedure for Alt-Ergo and a theory for Why3.

Miscellaneous

- Formalization of real analysis, with a focus on usability.
- Verification of a 3-point scheme for the wave equation.
- Safety invariant computation for hybrid systems.

Contributions

Floating-point arithmetic

- Formalization of high-level properties:
  rounding to odd, FMA, successor, etc.
- Formalization of IEEE-754 standard,
  used to implement a C compiler and verify its correctness.
- A procedure for Alt-Ergo and a theory for Why3.

Miscellaneous

- Formalization of real analysis, with a focus on usability.
- Verification of a 3-point scheme for the wave equation.
- Safety invariant computation for hybrid systems.
- Procedure for linear integer arithmetic.

Contributions

Floating-point arithmetic

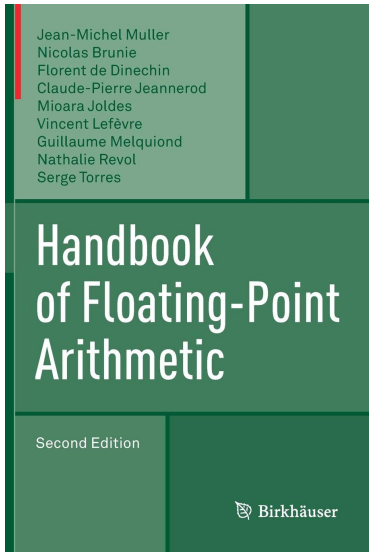
- Formalization of high-level properties:
  rounding to odd, FMA, successor, etc.
- Formalization of IEEE-754 standard,
  used to implement a C compiler and verify its correctness.
- A procedure for Alt-Ergo and a theory for Why3.

Miscellaneous

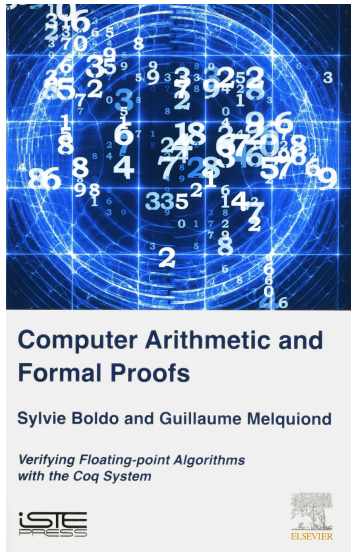
- Formalization of real analysis, with a focus on usability.
- Verification of a 3-point scheme for the wave equation.
- Safety invariant computation for hybrid systems.
- Procedure for linear integer arithmetic.
- Disproval of Masser-Gramain's conjecture: $\sum_k 1/(\pi r_k^2) = \ldots$

Advertisement

- Handbook of Floating-Point Arithmetic, Muller et al., 2010, 2018.



- Computer Arithmetic and Formal Proofs, Boldo, M., 2017.



What is Next?

Proof assistants and mathematics

Make the Coq environment as friendly to use
as computer algebra systems.

What is Next?

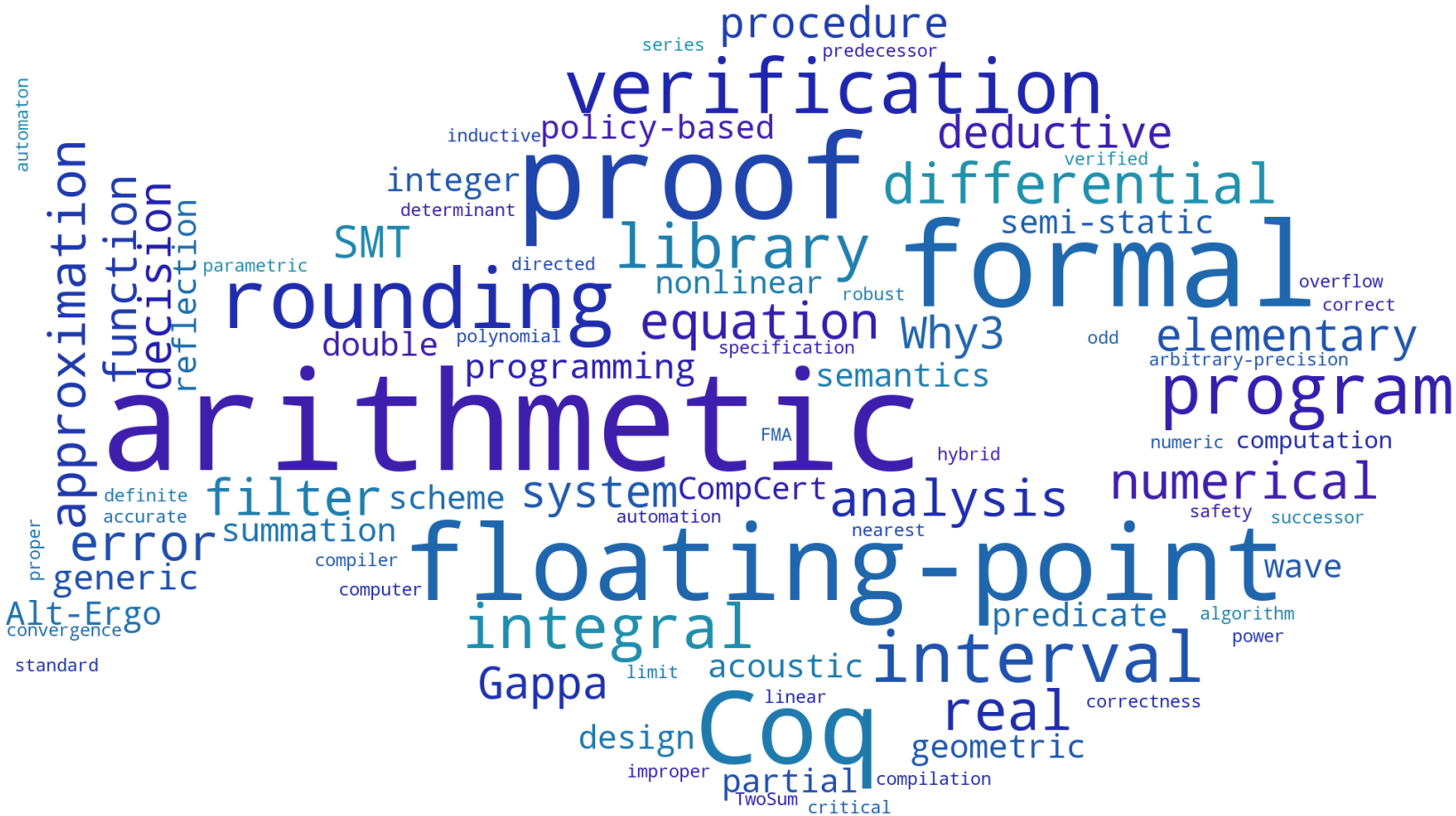
Proof assistants and mathematics

Make the Coq environment as friendly to use
as computer algebra systems.

Deductive program verification for the masses

Make Why3 and related tools a natural choice
to verify algorithms and libraries,
especially when it comes to floating-point code.

Question Time



Details: Computing tan

Approximation of $\tan x$ for $x \in [-\frac{1}{2}; \frac{1}{2}]$

1. Formally prove $\tan x = \dfrac{\sin x}{\sqrt{1 - \sin^2 x}}$,
2. and $0 \leq (-1)^n \left(\dfrac{\sin x}{x} - \displaystyle\sum_{k=0}^{n-1} (-1)^k \frac{x^{2k}}{(2k+1)!} \right) \leq \dfrac{x^{2n}}{(2n+1)!}$.
3. Implement $\tan : \mathbb{F} \to \mathbb{I}$ using extensions of $+$, $-$, $\times$, $\div$, $\sqrt{\bullet}$.
4. For $x \notin [-\frac{1}{2}; \frac{1}{2}]$, prove and implement an argument reduction.

Interval extension of tan

$$\forall x \in [\underline{x}; \overline{x}] \subset (-\tfrac{\pi}{2}; \tfrac{\pi}{2}), \quad \tan x \in \mathrm{hull}(\tan \underline{x}, \tan \overline{x}).$$

Details: Computing tan

Polynomial approximation of tan (also known as Taylor model)

1. Formally prove $\forall x \in \mathbf{x},\ \exists \xi \in \mathbf{x}$,
$$\tan x = \sum_{k=0}^{n} \frac{\tan^{(k)} x_0}{k!}(x - x_0)^k + \frac{\tan^{(n+1)} \xi}{(n+1)!}(x - x_0)^{n+1},$$
2. and $\dfrac{\tan^{(k)} u}{k!} = q_k(\tan u)$ with $q_{i+1} = \frac{1}{i+1}(q_i' + X^2 q_i') \in \mathbb{R}[X]$.
3. Implement the enclosure $(\vec{\mathbf{p}}, \boldsymbol{\Delta})$ of tan over $\mathbf{x}$ as $\vec{\mathbf{p}} = (\mathbf{q}_0(\tan x_0), \ldots, \mathbf{q}_n(\tan x_0))$ and $\boldsymbol{\Delta} = \mathbf{q}_{n+1}(\tan \mathbf{x}) \cdot \ldots$ using the interval extensions of q_i and tan.

Remark

Computing a Taylor model is a lot more efficient when the function has a linear differential equation, e.g., exp, sin, etc.

Automating the Error Analysis

Objective

Formally prove that,
if $X_0 = \tilde{x}_0 \cdot 2^{-8}$ approximates $rsa = 1/\sqrt{a}$ with 8 bits of accuracy,
then $X_1 = \tilde{x}_1 \cdot 2^{-64}$ approximates rsa with 16 bits of accuracy.

Gappa script

```
X1 = X0 + fixed<-24,dn>((X0 + fixed<-33,dn>(
  1 - 3b-33 - fixed<-33,dn>(a) * (X0 * X0))) * 0.5);
rsa = 1 / sqrt(a);
{ a in [0.25,1] /\ |(X0 - rsa) / X0| <= 1b-8 ->
  (X1 - rsa) / X1 in ? }
```

- Implicit quantification on free variables: $\forall a, X_0 \in \mathbb{R}$.
- Arithmetic operators on real numbers.
- Rounding operators: `fixed<`k`,dn>`$(x) = \lfloor x \cdot 2^{-k} \rfloor \cdot 2^k$.

Reflection for Why3, in a Nutshell

When everything else has failed...

If the current verification condition is $Q(y)$,
and if one has already verified the following WhyML function

```
let prove (x: t): bool
  requires { P x }
  ensures  { result = True -> Q x }
= ...
```

then one just needs to prove that $P(y)$ holds
and to check that `(prove y)` evaluates to `True`.

Remarks

- `prove` can make use of effects: mutability, exceptions, etc.
- If the current VC is not of the form $Q(y)$,
  Why3 uses reification to guess a suitable value for x.