Automations for Verifying Floating-point Algorithms in Coq

Guillaume Melquiond

Inria Saclay–Île-de-France

LRI, Université Paris Sud, CNRS

2013-07-22

Why Floating-point Arithmetic?

The real world is much more continuous than one could hope, so real numbers tend to creep in all the applications.

Why Floating-point Arithmetic?

The real world is much more continuous than one could hope, so real numbers tend to creep in all the applications.

How to compute with them?

- Use a subset, e.g. rational or algebraic numbers.

Why Floating-point Arithmetic?

The real world is much more continuous than one could hope, so real numbers tend to creep in all the applications.

How to compute with them?

- Use a subset, e.g. rational or algebraic numbers.
- Compute with arbitrary precision.

Why Floating-point Arithmetic?

The real world is much more continuous than one could hope, so real numbers tend to creep in all the applications.

How to compute with them?

- Use a subset, e.g. rational or algebraic numbers.
- Compute with arbitrary precision.
- Approximate operations, e.g. floating-point numbers.

Why Floating-point Arithmetic?

The real world is much more continuous than one could hope, so real numbers tend to creep in all the applications.

How to compute with them?

- Use a subset, e.g. rational or algebraic numbers.
- Compute with arbitrary precision.
- Approximate operations, e.g. floating-point numbers.

Speed of FP operations is high and deterministic,
but all bets are off with respect to the quality of FP results:
precision is known, but accuracy is not.

Why is FP Arithmetic Amenable to Formal Proof?

IEEE-754 standard for FP arithmetic

Every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result.

Why is FP Arithmetic Amenable to Formal Proof?

IEEE-754 standard for FP arithmetic

Every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result.

- Concise specification, suitable for program verification.

Why is FP Arithmetic Amenable to Formal Proof?

IEEE-754 standard for FP arithmetic

Every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result.

- Concise specification, suitable for program verification.
- It is all about real numbers.

Tutorial: FP Algorithms and Proof Automation

What kind of proof automation can we expect?

Tutorial: FP Algorithms and Proof Automation

What kind of proof automation can we expect?

Nothing new today, all the tools are at least 5-year old.

Tutorial: FP Algorithms and Proof Automation

What kind of proof automation can we expect?

Nothing new today, all the tools are at least 5-year old.

Example (FP algorithms and their Coq proofs)

1. Approximate the sine function:
   a straightforward proof about method and round-off errors.

Tutorial: FP Algorithms and Proof Automation

What kind of proof automation can we expect?

Nothing new today, all the tools are at least 5-year old.

Example (FP algorithms and their Coq proofs)

1. Approximate the sine function:
   a straightforward proof about method and round-off errors.
2. Perform an integer division:
   an intricate proof about convergent computations and exclusion zones.

Outline

Exceptional Values

Floating-point computations can lead to exceptional behaviors:

- invalid operations: $\sqrt{-1}$,
- overflow: $2 \times 2 \times \cdots \times 2$.

Exceptional Values

Floating-point computations can lead to exceptional behaviors:

- invalid operations: $\sqrt{-1}$,
- overflow: $2 \times 2 \times \cdots \times 2$.

When proving a FP algorithm, the very first step is to prove that

- exceptional behaviors cannot arise, or
- they are properly handled.

Exceptional Values

Floating-point computations can lead to exceptional behaviors:

- invalid operations: $\sqrt{-1}$,
- overflow: $2 \times 2 \times \cdots \times 2$.

When proving a FP algorithm, the very first step is to prove that

- exceptional behaviors cannot arise, or
- they are properly handled.

Today's talk is not about floating-point exceptions.
Let us assume that they are proved not to occur.

Floating-point Numbers and Real Numbers

Since there are no exceptional behaviors,
floating-point numbers can be embedded into real numbers.

Floating-point Numbers and Real Numbers

Since there are no exceptional behaviors,
floating-point numbers can be embedded into real numbers.

Representable numbers

$$\mathbb{F} = \{m \cdot \beta^e \in \mathbb{R} \mid m, e \in \mathbb{Z} \wedge |m| < \beta^p \wedge e \geq e_{\min}\}$$

with β, p, and $e_{\min}$ depending on the format.

Floating-point Numbers and Real Numbers

Since there are no exceptional behaviors,
floating-point numbers can be embedded into real numbers.

Representable numbers

$$\mathbb{F} = \{m \cdot \beta^e \in \mathbb{R} \mid m, e \in \mathbb{Z} \land |m| < \beta^p \land e \geq e_{\min}\}$$

with β, p, and $e_{\min}$ depending on the format.

Rounding operators

The result of an addition $a \oplus b$ is $\circ(a + b)$
with $\circ : \mathbb{R} \to \mathbb{F}$ a monotonic function that is the identity on $\mathbb{F}$.
$\circ(\cdot)$ depends on the destination format and the rounding direction.

Tools and Libraries

- Flocq: Coq formalization of floating-point arithmetic (any radix, any format).

Tools and Libraries

- Flocq: Coq formalization of floating-point arithmetic (any radix, any format).
- Gappa: C++ program for proving arithmetic properties involving rounding operators.

Tools and Libraries

- Flocq: Coq formalization of floating-point arithmetic (any radix, any format).
- Gappa: C++ program for proving arithmetic properties involving rounding operators.
- Interval: Coq tactic for proving bounds on differentiable real-valued expressions.

Interval Arithmetic

Interval arithmetic extends operations on real numbers to operations on closed connected subsets of real numbers.

Application

Instead of proving $\forall x \in [a, b],\ f(x) \in [c, d]$,
you can prove $F([a, b]) \subseteq [c, d]$,
assuming that F is an interval extension of f.

Interval Arithmetic

Interval arithmetic extends operations on real numbers to operations on closed connected subsets of real numbers.

Application

Instead of proving $\forall x \in [a, b],\ f(x) \in [c, d]$,
you can prove $F([a, b]) \subseteq [c, d]$,
assuming that F is an interval extension of f.

Evaluating F is easy; it involves operations on bounds only:

$$x \in [a, b] \wedge y \in [c, d] \Rightarrow x + y \in [a + c, b + d].$$

This makes interval arithmetic suitable for automatically proving bounds on real-valued expressions.

Outline

1. Introduction
2. Preliminaries
3. A straightforward example: sine around zero
   - Implementation
   - Method and round-off errors
   - Coq proof
   - The `interval` tactic
4. An intricate example: integer division
5. Conclusion

Example: Sine Around Zero

How to efficiently compute $\sin x$ for $|x| \le 1$
with a relative accuracy bounded by $103 \cdot 2^{-16}$?

Example: Sine Around Zero

How to efficiently compute $\sin x$ for $|x| \leq 1$
with a relative accuracy bounded by $103 \cdot 2^{-16}$?

Example (Toy sine)

```
float toy_sin(float x) {
  if (fabsf(x) < 0x1p-5f) return x;
  return x * (1.0f - x * x * 0x28e9p-16f);
}
```

Example: Sine Around Zero

How to efficiently compute $\sin x$ for $|x| \leq 1$
with a relative accuracy bounded by $103 \cdot 2^{-16}$?

Example (Toy sine)

```
float toy_sin(float x) {
  if (fabsf(x) < 0x1p-5f) return x;
  return x * (1.0f - x * x * 0x28e9p-16f);
}
```

An actual implementation of sin would

- use more than just 2 polynomials, and/or
- perform an argument reduction.

But the proof process is the same!

Approximating a Mathematical Function

How to compute an accurate FP approximation of $g(x)$ for any x?

Approximating a Mathematical Function

How to compute an accurate FP approximation of $g(x)$ for any x?

1. Find an approximation $\hat{g}$ of g that uses only real operations that can be approximated by your floating-point unit.

   Bound the method error $\varepsilon_m \geq |\hat{g}(x)/g(x) - 1|$.

Approximating a Mathematical Function

How to compute an accurate FP approximation of $g(x)$ for any x?

1. Find an approximation $\hat{g}$ of g that uses only real operations that can be approximated by your floating-point unit.
   Bound the method error $\varepsilon_m \geq |\hat{g}(x)/g(x) - 1|$.

2. Write $\tilde{g}$ that implements $\hat{g}$ with floating-point operations.
   Bound the round-off error $\varepsilon_r \geq |\tilde{g}(x)/\hat{g}(x) - 1|$.

Approximating a Mathematical Function

How to compute an accurate FP approximation of $g(x)$ for any x?

1. Find an approximation $\hat{g}$ of g that uses only real operations that can be approximated by your floating-point unit.
   Bound the method error $\varepsilon_m \geq |\hat{g}(x)/g(x) - 1|$.
2. Write $\tilde{g}$ that implements $\hat{g}$ with floating-point operations.
   Bound the round-off error $\varepsilon_r \geq |\tilde{g}(x)/\hat{g}(x) - 1|$.
3. Compose both bounds to get $\varepsilon \geq |\tilde{g}(x)/g(x) - 1|$.

Approximating a Mathematical Function

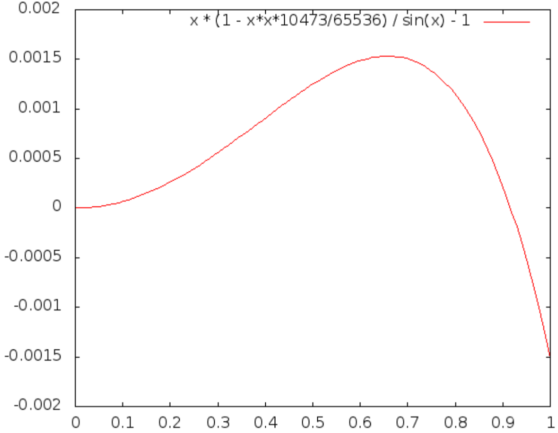
How to compute an accurate FP approximation of $g(x)$ for any x?

1. Find an approximation $\hat{g}$ of g that uses only real operations that can be approximated by your floating-point unit.
   Bound the method error $\varepsilon_m \geq |\hat{g}(x)/g(x) - 1|$.
2. Write $\tilde{g}$ that implements $\hat{g}$ with floating-point operations.
   Bound the round-off error $\varepsilon_r \geq |\tilde{g}(x)/\hat{g}(x) - 1|$.
3. Compose both bounds to get $\varepsilon \geq |\tilde{g}(x)/g(x) - 1|$.

Proving correctness is just a matter of computing tight bounds for these expressions.

Method Error (Relative)

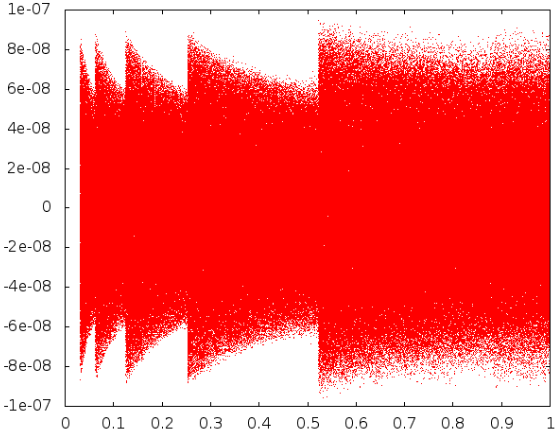
Method error: $\frac{x \cdot (1 - x^2 \cdot 10473 \cdot 2^{-16})}{\sin x} - 1.$



Tactic `interval` knows how to bound such an expression.

Binary32 Round-off Error (Relative)

Round-off error: $\frac{\circ(x \cdot \circ(1 - \circ(\circ(x^2) \cdot 10473 \cdot 2^{-16})))}{x \cdot (1 - x^2 \cdot 10473 \cdot 2^{-16})} - 1.$



Tactic gappa knows how to bound such an expression.
(And how to compose method and round-off errors.)

Correctness Statement in Coq

```
Notation fsub x y :=
  (round radix2 binary32_fmt rndNE (x - y)).
Notation fmul x y :=
  (round radix2 binary32_fmt rndNE (x * y)).

Definition fsin x :=
  if Rle_lt_dec (pow2 (-5)) (Rabs x) then
    fmul x (fsub 1 (fmul (fmul x x)
      (10473 * pow2 (-16))))
  else x.

Lemma sine_spec : forall x, Rabs x <= 1 ->
  Rabs (fsin x - sin x) <= 103*pow2 (-16) *
    Rabs (sin x).
```

Proof Sketch in Coq

```
Lemma sine_spec : forall x, Rabs x <= 1 ->
  Rabs (fsin x - sin x) <= 103 * pow2 (-16) *
    Rabs (sin x).
Proof.
intros x Bx. unfold fsin.
case Rle_lt_dec ; intros Bx'.
- (* |x| >= 1/32, degree-3 approx *)
  assert (Rabs (x * (1 - x * x * (10473*pow2 (-16))) -
       sin x) <= 102*pow2 (-16) * Rabs (sin x)).
    (* bound the method error *)
    interval with (i_bisect_diff x).
  (* bound the round-off and total errors *)
  gappa.
- (* |x| < 1/32, degree-1 approx *)
  destruct (MVT_cor2 sin cos).
  interval.
Qed.
```

What the Actual Coq Proof Looks Like

A Few Words About the `interval` Tactic

The scourge of interval arithmetic: the dependency effect.

Example

If $y \in [0,1]$, then $y - y \in [0-1, 1-0] = [-1, 1]$.
Impossible to prove $y - y = 0$ by interval arithmetic.

A Few Words About the `interval` Tactic

The scourge of interval arithmetic: the dependency effect.

Example

If $y \in [0,1]$, then $y - y \in [0-1, 1-0] = [-1, 1]$.
Impossible to prove $y - y = 0$ by interval arithmetic.

Note: the method error $\hat{g}(x) - g(x)$ shows such an effect.

A Few Words About the `interval` Tactic

The scourge of interval arithmetic: the dependency effect.

Example

If $y \in [0, 1]$, then $y - y \in [0 - 1, 1 - 0] = [-1, 1]$.
Impossible to prove $y - y = 0$ by interval arithmetic.

Note: the method error $\hat{g}(x) - g(x)$ shows such an effect.

`interval`

- "`interval`" performs naive interval arithmetic.

A Few Words About the `interval` Tactic

The scourge of interval arithmetic: the dependency effect.

Example

If $y \in [0, 1]$, then $y - y \in [0 - 1, 1 - 0] = [-1, 1]$.
Impossible to prove $y - y = 0$ by interval arithmetic.

Note: the method error $\hat{g}(x) - g(x)$ shows such an effect.

`interval`

- "`interval`" performs naive interval arithmetic.
- "`with (i_bisect x)`" subdivides the input range of `x`.

A Few Words About the `interval` Tactic

The scourge of interval arithmetic: the dependency effect.

Example

If $y \in [0,1]$, then $y - y \in [0-1, 1-0] = [-1, 1]$.
Impossible to prove $y - y = 0$ by interval arithmetic.

Note: the method error $\hat{g}(x) - g(x)$ shows such an effect.

`interval`

- "`interval`" performs naive interval arithmetic.
- "`with (i_bisect x)`" subdivides the input range of `x`.
- "`with (i_bisect_diff x)`" subdivides and applies order-1 arithmetic: $\forall x \in X,\ f(x) \in f(x_0) + (X - x_0) \times f'(X)$.

Outline

Integer Division on Itanium

Intel Itanium processors have no hardware divisor.
How to efficiently perform a division with just add and mul?

Integer Division on Itanium

Intel Itanium processors have no hardware divisor.
How to efficiently perform a division with just add and mul?

Example (Division of 16-bit unsigned integers on Itanium)

```
// Inputs:  dividend a in f6, divisor b in f7, 1+2^-17 in f9
      frcpa.s1   f8,p6=f6,f7 ;;
(p6) fma.s1      f6=f6,f8,f0
(p6) fnma.s1     f7=f7,f8,f9 ;;
(p6) fma.s1      f8=f7,f6,f6 ;;
      fcvt.fx.trunc.s1 f8=f8
// Output:  ⌊a/b⌋ in f8
```

Integer Division on Itanium

Intel Itanium processors have no hardware divisor.
How to efficiently perform a division with just add and mul?

Example (Division of 16-bit unsigned integers on Itanium)

```
// Inputs:  dividend a in f6, divisor b in f7, 1+2^-17 in f9
      frcpa.s1    f8,p6=f6,f7 ;;
(p6) fma.s1       f6=f6,f8,f0
(p6) fnma.s1      f7=f7,f8,f9 ;;
(p6) fma.s1       f8=f7,f6,f6 ;;
      fcvt.fx.trunc.s1 f8=f8
// Output:  ⌊a/b⌋ in f8
```

- Cornea, Iordache, Harrison, Markstein, "Integer Divide and Remainder Operations in the Intel IA-64 Architecture," RNC 2000.
- Harrison, "Formal verification of IA-64 division algorithms," TPHOL 2000.

Integer Division on Itanium

Example (Division of 16-bit unsigned integers on Itanium)

$$\begin{aligned} y_0 &\approx 1/b \qquad [\texttt{frcpa}] \\ q_0 &= \circ(a \times y_0) \\ e_0 &= \circ(1 + 2^{-17} - b \times y_0) \\ q_1 &= \circ(e_0 \times q_0 + q_0) \\ q &= \lfloor q_1 \rfloor \end{aligned}$$

with $\circ(\cdot)$ rounding to nearest on the extended 82-bit format.

Correctness of the division

$$\forall a, b \in [\![1; 65535]\!], \quad q = \lfloor a/b \rfloor.$$

Correctness Statement in Coq

```
Notation fma x y z :=
  (round radix2 register_fmt rndNE (x * y + z)).

Axiom frcpa : R -> R.
Axiom frcpa_spec : forall x : R,
  1 <= Rabs x <= 65536 ->
  generic_format radix2 (FLT_exp _ 11) (frcpa x) /\
  Rabs (frcpa x - 1/x) <= 4433*pow2 (-21) * Rabs(1/x).

Definition div_u16 a b :=
  let y0 := frcpa b in
  let q0 := fma a y0 0 in
  let e0 := fnma b y0 (1 + pow2 (-17)) in
  let q1 := fma e0 q0 q0 in
  Zfloor q1.

Lemma div_u16_spec : forall a b,
  (1 <= a <= 65535)%Z ->
  (1 <= b <= 65535)%Z ->
  div_u16 a b = (a / b)%Z.
```

Proof Sketch

Theorem (Exclusion zones)

Given a and b positive integers.
If $0 \leq a \times (q_1/(a/b) - 1) < 1$, *then* $\lfloor q_1 \rfloor = \lfloor a/b \rfloor$.

Proof Sketch

Theorem (Exclusion zones)

Given a and b positive integers.
If $0 \leq a \times (q_1/(a/b) - 1) < 1$, *then* $\lfloor q_1 \rfloor = \lfloor a/b \rfloor$.

Proof.

By equivalence between the following properties:

1. $\lfloor a/b \rfloor \leq q_1 < \lfloor a/b \rfloor + 1$.
2. $b \times \lfloor a/b \rfloor - a \leq b \times q_1 - a < b \times (\lfloor a/b \rfloor + 1) - a$.
3. $-(a \bmod b) \leq a \times (q_1/(a/b) - 1) < b - (a \bmod b)$.

□

Proof Sketch

Theorem (Exclusion zones)

Given a and b positive integers.
If $0 \le a \times (q_1/(a/b) - 1) < 1$, *then* $\lfloor q_1 \rfloor = \lfloor a/b \rfloor$.

Proof.

By equivalence between the following properties:

1. $\lfloor a/b \rfloor \le q_1 < \lfloor a/b \rfloor + 1$.
2. $b \times \lfloor a/b \rfloor - a \le b \times q_1 - a < b \times (\lfloor a/b \rfloor + 1) - a$.
3. $-(a \bmod b) \le a \times (q_1/(a/b) - 1) < b - (a \bmod b)$.

□

Notice the relative error between the FP value q_1 and the real a/b.
So proving the correctness is just a matter of bounding this error.

Proof Sketch Continued

Bounding the method error $\hat{q}_1 - a/b$ and the round-off error $q_1 - \hat{q}_1$ and composing them does not work at all.

Proof Sketch Continued

Bounding the method error $\hat{q}_1 - a/b$ and the round-off error $q_1 - \hat{q}_1$ and composing them does not work at all.

What the developers knew when designing the algorithm:

- If not for 2^{-17}, the code would perform a Newton iteration: $\hat{q}_1/(a/b) - 1 = -\varepsilon_0^2$ with $\varepsilon_0 = y_0/(1/b) - 1$.
- By taking into account 2^{-17}, $\hat{q}_1/(a/b) - 1 = -\varepsilon_0^2 + (1 + \varepsilon_0) \cdot 2^{-17}$.

Proof Sketch, the Coq Version

```
Lemma div_u16_spec : forall a b,
  (1 <= a <= 65535)%Z -> (1 <= b <= 65535)%Z ->
  div_u16 a b = (a / b)%Z.
Proof.
intros a b Ba Bb.
apply Zfloor_imp.
cut (0 <= b * q1 - a < 1).
  lra.
set (err := (q1 - a / b) / (a / b)).
replace (b * q1 - a) with (a * err) by field.
set (y0 := frcpa b).
set (Mq0 := a * y0 + 0).
set (Me0 := 1 + pow2 (-17) - b * y0).
set (Mq1 := Me0 * Mq0 + Mq0).
set (eps0 := (y0 - 1 / b) / (1 / b)).
assert ((Mq1 - a / b) / (a / b) =
  -(eps0 * eps0) + (1 + eps0) * pow2 (-17)) by field.
generalize (frcpa_spec b) (FIX_format_Z2R radix2 a)
  (FIX_format_Z2R radix2 b).
gappa.
Qed.
```

What the Actual Coq Proof Looks Like

A few Words About the gappa Tactic

Starting from a formula, Gappa saturates a set of theorems to deduce new properties until it encounters a contradiction.

A few Words About the gappa Tactic

Starting from a formula, Gappa saturates a set of theorems to deduce new properties until it encounters a contradiction.

Supported properties

$$
\begin{array}{lcll}
\texttt{BND}(x, I) & \equiv & x \in I & \\
\texttt{ABS}(x, I) & \equiv & |x| \in I & \\
\texttt{REL}(x, y, I) & \equiv & \exists \varepsilon \in I, & x = y \cdot (1 + \varepsilon) \\
\texttt{FIX}(x, e) & \equiv & \exists m \in \mathbb{Z}, & x = m \cdot 2^e \\
\texttt{FLT}(x, p) & \equiv & \exists m, e \in \mathbb{Z}, & x = m \cdot 2^e \wedge |m| < 2^p \\
\texttt{NZR}(x) & \equiv & x \neq 0 & \\
\texttt{EQL}(x, y) & \equiv & x = y &
\end{array}
$$

A few Words About the gappa Tactic

Starting from a formula, Gappa saturates a set of theorems to deduce new properties until it encounters a contradiction.

Supported properties

$$
\begin{array}{lcll}
\mathtt{BND}(x, I) & \equiv & x \in I & \\
\mathtt{ABS}(x, I) & \equiv & |x| \in I & \\
\mathtt{REL}(x, y, I) & \equiv & \exists \varepsilon \in I, & x = y \cdot (1 + \varepsilon) \\
\mathtt{FIX}(x, e) & \equiv & \exists m \in \mathbb{Z}, & x = m \cdot 2^e \\
\mathtt{FLT}(x, p) & \equiv & \exists m, e \in \mathbb{Z}, & x = m \cdot 2^e \wedge |m| < 2^p \\
\mathtt{NZR}(x) & \equiv & x \neq 0 & \\
\mathtt{EQL}(x, y) & \equiv & x = y &
\end{array}
$$

On the example, Gappa tries to apply about 2000 theorems.
The final proof manipulates about 100 properties.

Where Does the Specification of `frcpa` Come From?

How do we know $|\varepsilon_0| \leq 4433 \cdot 2^{-21}$ and that y_0 fits on 11 bits?
By reading the pseudo-code:

```
fp_ieee_recip(den)
{
  RECIP_TABLE[256] = {
    0x3fc,0x3f4,0x3ec,0x3e4,0x3dd,0x3d5,0x3cd,0x3c6,
    // ... 29 lines ...
    0x020,0x01e,0x01c,0x01a,0x018,0x015,0x013,0x011,
    0x00f,0x00d,0x00b,0x009,0x007,0x005,0x003,0x001,
  };

  tmp_index = den.significand{62:55};
  tmp_res.significand = (1 << 63) | (RECIP_TABLE[
      tmp_index] << 53);
  tmp_res.exponent = FP_REG_EXP_ONES - 2 - den.
      exponent;
  tmp_res.sign = den.sign;
  return (tmp_res);
}
```

Correctness of frcpa

```
Definition recip_table :=
  2044::2036::2028::2020::2013::2005::1997::1990::
  1982::1975::1967::1960::1953::1945::1938::1931::
  ...

Lemma frcpa_spec : forall i x,
  (0 <= i < 256)%nat ->
  INR (256 + i)/256 <= x <= INR (256 + S i)/256 ->
  Rabs (nth i recip_table 0 / 2048 - 1 / x) <=
    4433 * pow2 (-21) * Rabs (1 / x).
Proof.
intros i x Bi Bx.
destruct (le_eq_or_S _ _ (proj1 Bi)).
  interval.
destruct (le_eq_or_S _ _ (proj1 Bi)).
  interval.
(* ... repeat 254 more times *)
Qed.
```

Outline

Conclusion

- Gappa supports:
  - arithmetic operators: $+$, $\times$, $\sqrt{\cdot}$,
  - rounding operators for fixed- and floating-point numbers,
  - constraints and algebraic relations.

Conclusion

- Gappa supports:
  - arithmetic operators: $+$, $\times$, $\sqrt{\cdot}$,
  - rounding operators for fixed- and floating-point numbers,
  - constraints and algebraic relations.
- Interval supports:
  - elementary functions: cos, arctan, exp,
  - order-1 interval arithmetic.

Conclusion

- Gappa supports:
  - arithmetic operators: $+$, $\times$, $\sqrt{\cdot}$,
  - rounding operators for fixed- and floating-point numbers,
  - constraints and algebraic relations.
- Interval supports:
  - elementary functions: cos, arctan, exp,
  - order-1 interval arithmetic.
- Issues:
  - verifying Gappa-generated proofs is slow;
  - order-1 IA is not enough for some applications.

Questions?

Flocq:	`http://flocq.gforge.inria.fr/`
Gappa:	`http://gappa.gforge.inria.fr/`
Interval:	`https://www.lri.fr/~melquion/soft/coq-interval/`