

# Gappa: Verifying Properties of Floating-Point Code

Guillaume Melquiond

January 22, 2026

# Upsides and Downsides of Computer Arithmetic, Example 1

## Example (GMP's 64-bit integer square root: `mpn_sqrtrem1`)

```

abits = a0 >> (64-8-1);
x0 = 0x100 | invsqrttab[abits - 0x80];
a1 = a0 >> (64-1-32);
t = (signed) (0x2000000000000000 - 0x30000 - a1 * x0 * x0) >> 16;
x0 = (x0 << 16) + ((signed) (x0 * t) >> (16+2));
t2 = x0 * (a0 >> (32-8));
t = t2 >> 25;
t = ((signed) ((a0 << 14) - t * t - MAGIC) >> (32-8));
x0 = t2 + ((signed) (x0 * t) >> 15);
x0 >>= 32;
x2 = x0 * x0;
if (x2 + 2*x0 <= a0 - 1) { x2 += 2*x0 + 1; x0++; }

```

/\* extract bits for table lookup \*/  
/\* x0 is now a 8 bits approximation of 1/sqrt(a0) \*/  
/\* x0 is now a 16 bits approximation of 1/sqrt(a0) \*/  
/\* x0 is now a full limb approximation of sqrt(a0) \*/

Specification:  $2^{62} \leq a_0 < 2^{64} \Rightarrow x_0^2 \leq a_0 < (x_0 + 1)^2$ .

# Upsides and Downsides of Computer Arithmetic, Example 1

## Example (GMP's 64-bit integer square root: `mpn_sqrtrem1`)

```

abits = a0 >> (64-8-1);
x0 = 0x100 | invsqrttab[abits - 0x80];
a1 = a0 >> (64-1-32);
t = (signed) (0x20000000000000 - 0x30000 - a1 * x0 * x0) >> 16;
x0 = (x0 << 16) + ((signed) (x0 * t) >> (16+2));
t2 = x0 * (a0 >> (32-8));
t = t2 >> 25;
t = ((signed) ((a0 << 14) - t * t - MAGIC) >> (32-8));
x0 = t2 + ((signed) (x0 * t) >> 15);
x0 >>= 32;
x2 = x0 * x0;
if (x2 + 2*x0 <= a0 - 1) { x2 += 2*x0 + 1; x0++; }

```

/\* extract bits for table lookup \*/  
/\* x0 is now a 8 bits approximation of 1/sqrt(a0) \*/  
/\* x0 is now a 16 bits approximation of 1/sqrt(a0) \*/  
/\* x0 is now a full limb approximation of sqrt(a0) \*/

Specification:  $2^{62} \leq a_0 < 2^{64} \Rightarrow x_0^2 \leq a_0 < (x_0 + 1)^2$ .

- Implementations are short yet intricate.
- Specifications are precise yet simple.
- Input domains are often too large for exhaustive testing.
- Correctness proofs are often unreadable, assuming they even exist.

# Upsides and Downsides of Computer Arithmetic, Example 2

## Example (Double-double multiplication)

```
uh = xh * yh;  
ul = fma(xh, yh, -uh);  
ul += xh * yl + xl * yh;  
zh = uh + ul;  
zl = zh - uh - ul;
```

Specification: In the absence of under/overflow,

$$x_h = \circ(x_h + x_l) \wedge y_h = \circ(y_h + y_l) \Rightarrow |(z_h + z_l) / ((x_h + x_l) \cdot (y_h + y_l)) - 1| \leq 5 \cdot 2^{-106}.$$

# Upsides and Downsides of Computer Arithmetic, Example 2

## Example (Double-double multiplication)

```
uh = xh * yh;  
ul = fma(xh, yh, -uh);  
ul += xh * yl + xl * yh;  
zh = uh + ul;  
zl = zh - uh - ul;
```

Specification: In the absence of under/overflow,

$$x_h = \circ(x_h + x_l) \wedge y_h = \circ(y_h + y_l) \Rightarrow |(z_h + z_l) / ((x_h + x_l) \cdot (y_h + y_l)) - 1| \leq 5 \cdot 2^{-106}.$$

- Implementations often seem nonsensical.
- Specifications are often difficult to test.
- Correctness proofs often use very arcane arguments.

# How to Help Experts Verify their State-of-the-Art Code?

## Gappa

### Design decisions

- Targeted at intricate use cases.
- With a high confidence in the results.
- Very fast, for an “interactive” use.
- Meant for experts with lots of domain-specific knowledge.

# Gappa: an Automated Prover for Computer Arithmetic Properties

## Targeted properties

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

# Gappa: an Automated Prover for Computer Arithmetic Properties

## Targeted properties

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

## Example (Not quite Sterbenz)

$$x \in [1; 3] \wedge y \in [1; 3] \Rightarrow \circ(\circ(x) - \circ(y)) - (\circ(x) - \circ(y)) \in [0; 0]$$

with  $\circ$  rounding to nearest, tie breaking to even, in binary64.

# Gappa: an Automated Prover for Computer Arithmetic Properties

## Targeted properties

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

## Example (Not quite Sterbenz)

$x \in [1; 3] \wedge y \in [1; 3] \Rightarrow \circ(\circ(x) - \circ(y)) - (\circ(x) - \circ(y)) \in [0; 0]$

with  $\circ$  rounding to nearest, tie breaking to even, in binary64.

```
@r = float<ieee_64,ne>;
```

```
{ x in [1,3] /\ y in [1,3] -> r(r(x)-r(y)) - (r(x)-r(y)) in [0,0] }
```

# (Non) Features of Gappa

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

# (Non) Features of Gappa

Quantifier-free formulas composed of enclosures of

- **arithmetic expressions**  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

## Unsupported features

- No loops, no data structures.

# (Non) Features of Gappa

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

## Unsupported features

- No loops, no data structures.
- No transcendental functions.

# (Non) Features of Gappa

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

## Unsupported features

- No loops, no data structures.
- No transcendental functions.
- No exceptional behaviors.

# (Non) Features of Gappa

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with **rounding operators**  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

## Unsupported features

- No loops, no data structures.
- Not agnostic to precision or direction.
- No transcendental functions.
- No exceptional behaviors.

# (Non) Features of Gappa

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with **numerical bounds**.

## Unsupported features

- No loops, no data structures.
- No transcendental functions.
- No exceptional behaviors.
- Not agnostic to precision or direction.
- No fancy symbolic properties.

# (Non) Features of Gappa

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

## Unsupported features

- No loops, no data structures.
- No transcendental functions.
- No exceptional behaviors.
- Not agnostic to precision or direction.
- No fancy symbolic properties.

## Supported features

- Fixed- and floating-point arithmetic.

# (Non) Features of Gappa

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

## Unsupported features

- No loops, no data structures.
- No transcendental functions.
- No exceptional behaviors.
- Not agnostic to precision or direction.
- No fancy symbolic properties.

## Supported features

- Fixed- and floating-point arithmetic.
- Generation of formal proofs.

# (Non) Features of Gappa

Quantifier-free formulas composed of enclosures of

- arithmetic expressions  $(+, -, \times, \div, \sqrt{\cdot})$  over  $\mathbb{R}$  with rounding operators  $(\mathbb{R} \rightarrow \mathbb{R})$
- inside intervals with numerical bounds.

## Unsupported features

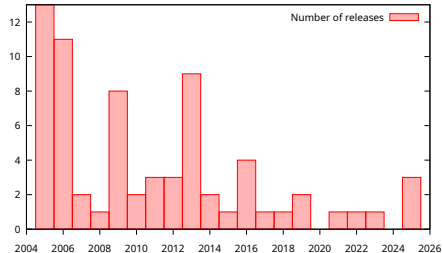
- No loops, no data structures.
- No transcendental functions.
- No exceptional behaviors.
- Not agnostic to precision or direction.
- No fancy symbolic properties.

## Supported features

- Fixed- and floating-point arithmetic.
- Generation of formal proofs.
- User-provided rules.

# Gappa: History

Started in late 2004. Release cycle mostly driven by user needs.



# Outline

- 1 Gappa's proof engine
- 2 The database of theorems
- 3 Examples
- 4 Conclusion

# Outline

- 1 Gappa's proof engine
  - Saturation-based reasoning
  - Supported predicates
  - Proof stages
  - Summary
- 2 The database of theorems
- 3 Examples
- 4 Conclusion

# Verifying Properties Using Saturation

- 1 Initialize the set of clauses using the **negation of the goal**.
- 2 Extend the set of clauses by **applying a rule** to a subset of the clauses.
- 3 If the **empty clause** was added, succeed. Otherwise, go back to step 2.

# Verifying Properties Using Saturation

- 1 Initialize the set of clauses using the **negation of the goal**.
- 2 Extend the set of clauses by **applying a rule** to a subset of the clauses.
- 3 If the **empty clause** was added, succeed. Otherwise, go back to step 2.

Example  $(x \in [1; 2] \Rightarrow x + 1 \in [0; 3])$

# Verifying Properties Using Saturation

- 1 Initialize the set of clauses using the **negation of the goal**.
- 2 Extend the set of clauses by **applying a rule** to a subset of the clauses.
- 3 If the **empty clause** was added, succeed. Otherwise, go back to step 2.

**Example**  $(x \in [1; 2] \Rightarrow x + 1 \in [0; 3])$

- 1  $\{\langle x \in [1; 2] \rangle, \langle x + 1 \notin [0; 3] \rangle\}$  initially.

# Verifying Properties Using Saturation

- 1 Initialize the set of clauses using the **negation of the goal**.
- 2 Extend the set of clauses by **applying a rule** to a subset of the clauses.
- 3 If the **empty clause** was added, succeed. Otherwise, go back to step 2.

**Example**  $(x \in [1; 2] \Rightarrow x + 1 \in [0; 3])$

- 1  $\{\langle x \in [1; 2] \rangle, \langle x + 1 \notin [0; 3] \rangle\}$  initially.
- 2  $\{\langle x \in [1; 2] \rangle, \langle x + 1 \notin [0; 3] \rangle, \langle x + 1 \in [2; 3] \rangle\}$  using interval addition.

# Verifying Properties Using Saturation

- 1 Initialize the set of clauses using the **negation of the goal**.
- 2 Extend the set of clauses by **applying a rule** to a subset of the clauses.
- 3 If the **empty clause** was added, succeed. Otherwise, go back to step 2.

**Example**  $(x \in [1; 2] \Rightarrow x + 1 \in [0; 3])$

- 1  $\{\langle x \in [1; 2] \rangle, \langle x + 1 \notin [0; 3] \rangle\}$  initially.
- 2  $\{\langle x \in [1; 2] \rangle, \langle x + 1 \notin [0; 3] \rangle, \langle x + 1 \in [2; 3] \rangle\}$  using interval addition.
- 3  $\{\langle x \in [1; 2] \rangle, \langle x + 1 \notin [0; 3] \rangle, \langle x + 1 \in [2; 3] \rangle, \langle x + 1 \in \emptyset \rangle\}$  using intersection.

# Verifying Properties Using Saturation

- 1 Initialize the set of clauses using the **negation of the goal**.
- 2 Extend the set of clauses by **applying a rule** to a subset of the clauses.
- 3 If the **empty clause** was added, succeed. Otherwise, go back to step 2.

**Example**  $(x \in [1; 2] \Rightarrow x + 1 \in [0; 3])$

- 1  $\{\langle x \in [1; 2] \rangle, \langle x + 1 \notin [0; 3] \rangle\}$  initially.
- 2  $\{\langle x \in [1; 2] \rangle, \langle x + 1 \notin [0; 3] \rangle, \langle x + 1 \in [2; 3] \rangle\}$  using interval addition.
- 3  $\{\langle x \in [1; 2] \rangle, \langle x + 1 \notin [0; 3] \rangle, \langle x + 1 \in [2; 3] \rangle, \langle x + 1 \in \emptyset \rangle\}$  using intersection.

Gappa is not refutationally complete.

# The Inference Rules of Gappa

## Supported clauses, abridged

- $\langle x \in X \rangle$  with  $x$  a **symbolic expression** and  $X$  an **interval with numerical bounds**.
- Some other atomic clauses. . .
- Arbitrary propositional formulas containing atomic clauses and their negation.

# The Inference Rules of Gappa

## Supported clauses, abridged

- $\langle x \in X \rangle$  with  $x$  a **symbolic expression** and  $X$  an **interval with numerical bounds**.
- Some other atomic clauses. . .
- Arbitrary propositional formulas containing atomic clauses and their negation.

## Example (Addition)

- Lemma:  $\forall U, V, W \in \mathbb{I}, \forall u, v \in \mathbb{R}, \quad u \in U \wedge v \in V \wedge U + V \subseteq W \Rightarrow u + v \in W.$

# The Inference Rules of Gappa

## Supported clauses, abridged

- $\langle x \in X \rangle$  with  $x$  a **symbolic expression** and  $X$  an **interval with numerical bounds**.
- Some other atomic clauses. . .
- Arbitrary propositional formulas containing atomic clauses and their negation.

## Example (Addition)

- Lemma:  $\forall U, V, W \in \mathbb{I}, \forall u, v \in \mathbb{R}, \quad u \in U \wedge v \in V \wedge U + V \subseteq W \Rightarrow u + v \in W$ .
- Input clauses:  $\langle u \in U \rangle, \langle v \in V \rangle$ . Output clause:  $\langle u + v \in W \rangle$ .
- Computational content:  $U + V \subseteq W$ .

# The Inference Rules of Gappa

## Supported clauses, abridged

- $\langle x \in X \rangle$  with  $x$  a **symbolic expression** and  $X$  an **interval with numerical bounds**.
- Some other atomic clauses. . .
- Arbitrary propositional formulas containing atomic clauses and their negation.

## Example (Addition)

- Lemma:  $\forall U, V, W \in \mathbb{I}, \forall u, v \in \mathbb{R}, \quad u \in U \wedge v \in V \wedge U + V \subseteq W \Rightarrow u + v \in W$ .
- Input clauses:  $\langle u \in U \rangle, \langle v \in V \rangle$ . Output clause:  $\langle u + v \in W \rangle$ .
- Computational content:  $U + V \subseteq W$ .
- Symbolic shorthand:  $\text{BND}(u + v) \Leftarrow \text{BND}(u) \wedge \text{BND}(v)$ .

## Aside: Interval Arithmetic in a Nutshell

Intervals: closed connected subsets of  $\mathbb{R}$

- Stored as a pair of floating-point numbers:  $\mathbb{I} = \mathbb{F}^2$ .
- Interpretation:  $x \in [\underline{x}, \overline{x}]$ .

## Aside: Interval Arithmetic in a Nutshell

Intervals: closed connected subsets of  $\mathbb{R}$

- Stored as a pair of floating-point numbers:  $\mathbb{I} = \mathbb{F}^2$ .
- Interpretation:  $x \in [\underline{x}, \overline{x}]$ .

How to compute  $W$  such that  $U + V \subseteq W$ ?

If  $u \in [\underline{u}, \overline{u}]$  and  $v \in [\underline{v}, \overline{v}]$ , then  $u + v \in [\nabla(\underline{u} + \underline{v}), \Delta(\overline{u} + \overline{v})]$ ,  
by monotonicity of addition and properties of directed rounding  $\nabla, \Delta$ .

## Aside: Interval Arithmetic in a Nutshell

Intervals: closed connected subsets of  $\mathbb{R}$

- Stored as a pair of floating-point numbers:  $\mathbb{I} = \mathbb{F}^2$ .
- Interpretation:  $x \in [\underline{x}, \overline{x}]$ .

How to compute  $W$  such that  $U + V \subseteq W$ ?

If  $u \in [\underline{u}, \overline{u}]$  and  $v \in [\underline{v}, \overline{v}]$ , then  $u + v \in [\nabla(\underline{u} + \underline{v}), \Delta(\overline{u} + \overline{v})]$ ,  
by monotonicity of addition and properties of directed rounding  $\nabla, \Delta$ .

Gappa's specifics

- Precision by default: 60 bits (GNU MPFR).
- Intervals with infinite bounds are second-class citizens.

# Supported Predicates

## Domain of expressions

- $\text{BND}(x, J) \equiv x \in J,$

# Supported Predicates

## Domain of expressions

- $\text{BND}(x, J) \equiv x \in J$ ,
- $\text{ABS}(x, J) \equiv |x| \in J$ , (e.g., normal number)
- $\text{NZR}(x) \equiv x \neq 0$ . (e.g., can divide)

# Supported Predicates

## Domain of expressions

- $\text{BND}(x, J) \equiv x \in J$ ,
- $\text{ABS}(x, J) \equiv |x| \in J$ , (e.g., normal number)
- $\text{NZR}(x) \equiv x \neq 0$ . (e.g., can divide)

## Limited precision

- $\text{FIX}(x, k) \equiv \exists m, e \in \mathbb{Z}, x = m \cdot 2^e \wedge e \geq k$ , (e.g., integer)
- $\text{FLT}(x, k) \equiv \exists m, e \in \mathbb{Z}, x = m \cdot 2^e \wedge |m| < 2^k$ . (e.g., result of a rounding)

# Supported Predicates

## Domain of expressions

- $\text{BND}(x, J) \equiv x \in J$ ,
- $\text{ABS}(x, J) \equiv |x| \in J$ , (e.g., normal number)
- $\text{NZR}(x) \equiv x \neq 0$ . (e.g., can divide)

## Limited precision

- $\text{FIX}(x, k) \equiv \exists m, e \in \mathbb{Z}, x = m \cdot 2^e \wedge e \geq k$ , (e.g., integer)
- $\text{FLT}(x, k) \equiv \exists m, e \in \mathbb{Z}, x = m \cdot 2^e \wedge |m| < 2^k$ . (e.g., result of a rounding)

## Relational properties

- $\text{REL}(x, y, J) \equiv \exists \varepsilon \in J, x = y \cdot (1 + \varepsilon)$ , (e.g., relative error)
- $\text{LIN}(x, y, J) \equiv \exists \varepsilon \in J, x = y \cdot \varepsilon$ , (e.g., multi-word arithmetic)

# Supported Predicates

## Domain of expressions

- $\text{BND}(x, J) \equiv x \in J$ ,
- $\text{ABS}(x, J) \equiv |x| \in J$ , (e.g., normal number)
- $\text{NZR}(x) \equiv x \neq 0$ . (e.g., can divide)

## Limited precision

- $\text{FIX}(x, k) \equiv \exists m, e \in \mathbb{Z}, x = m \cdot 2^e \wedge e \geq k$ , (e.g., integer)
- $\text{FLT}(x, k) \equiv \exists m, e \in \mathbb{Z}, x = m \cdot 2^e \wedge |m| < 2^k$ . (e.g., result of a rounding)

## Relational properties

- $\text{REL}(x, y, J) \equiv \exists \varepsilon \in J, x = y \cdot (1 + \varepsilon)$ , (e.g., relative error)
- $\text{LIN}(x, y, J) \equiv \exists \varepsilon \in J, x = y \cdot \varepsilon$ , (e.g., multi-word arithmetic)
- $\text{EQL}(x, y) \equiv x = y$ . (e.g., exact operation)

# Proof Stages

- ① Heuristically generate the inference rules from a database of theorems.
- ② Numerically saturate the clauses using the generated inference rules.
- ③ Simplify the trace and generate a formal proof. (optional)

# Proof Stages

- ① Heuristically generate the inference rules from a database of theorems.
- ② Numerically saturate the clauses using the generated inference rules.
- ③ Simplify the trace and generate a formal proof. (optional)

## The saturation stage

- Infer new clauses mostly by using naive interval arithmetic.
- Keep a trace of the applied theorems if a proof needs to be generated.

# Generating the Inference Rules, Part 1

## Backward reasoning by depth-first search

- ① Start from a clause present in the goal.
- ② Symbolically instantiate all the theorems that can produce this clause.
- ③ Recursively go to step 2 to visit all the hypotheses of these theorems.

# Generating the Inference Rules, Part 1

## Backward reasoning by depth-first search

- 1 Start from a clause present in the goal.
- 2 Symbolically instantiate all the theorems that can produce this clause.
- 3 Recursively go to step 2 to visit all the hypotheses of these theorems.

Example (Generated rules for  $x \in [1; 2] \Rightarrow x + 1 \in [0; 3]$ )

- $\text{BND}(x + 1) \Leftarrow \text{BND}(x) \wedge \text{BND}(1),$
- $\text{BND}(1),$

# Generating the Inference Rules, Part 1

## Backward reasoning by depth-first search

- 1 Start from a clause present in the goal.
- 2 Symbolically instantiate all the theorems that can produce this clause.
- 3 Recursively go to step 2 to visit all the hypotheses of these theorems.

## Example (Generated rules for $x \in [1; 2] \Rightarrow x + 1 \in [0; 3]$ )

- $\text{BND}(x + 1) \Leftarrow \text{BND}(x) \wedge \text{BND}(1),$
- $\text{BND}(1),$
- $\text{BND}(x) \Leftarrow \text{BND}((x + 1) - 1),$
- $\text{BND}((x + 1) - 1) \Leftarrow \text{BND}(x + 1) \wedge \text{BND}(1),$

# Generating the Inference Rules, Part 1

## Backward reasoning by depth-first search

- 1 Start from a clause present in the goal.
- 2 Symbolically instantiate all the theorems that can produce this clause.
- 3 Recursively go to step 2 to visit all the hypotheses of these theorems.

## Example (Generated rules for $x \in [1; 2] \Rightarrow x + 1 \in [0; 3]$ )

- $\text{BND}(x + 1) \Leftarrow \text{BND}(x) \wedge \text{BND}(1),$
- $\text{BND}(x) \Leftarrow \text{BND}((x + 1) - 1),$
- $\text{BND}((x + 1) - 1) \Leftarrow \text{BND}(x + 1) \wedge \text{BND}(1),$
- $\text{BND}(1),$
- $\text{BND}(x) \Leftarrow \text{ABS}(x),$
- 40 other rules.

## Generating the Inference Rules, Part 2

The trouble with  $\text{BND}(x) \Leftarrow \text{BND}((x + 1) - 1)$

Theorem:  $\forall J \in \mathbb{I}, \forall a, b \in \mathbb{R}, (a + b) - b \in J \Rightarrow a \in J.$

## Generating the Inference Rules, Part 2

The trouble with  $\text{BND}(x) \Leftarrow \text{BND}((x + 1) - 1)$

Theorem:  $\forall J \in \mathbb{I}, \forall a, b \in \mathbb{R}, (a + b) - b \in J \Rightarrow a \in J$ .

To prove  $\text{BND}(x)$ , there is only **one choice** for  $a$ :  $a = x$ .

But there are **infinitely many** choices for  $b$ . Why choose  $b = 1$ ?

## Generating the Inference Rules, Part 2

The trouble with  $\text{BND}(x) \Leftarrow \text{BND}((x + 1) - 1)$

Theorem:  $\forall J \in \mathbb{I}, \forall a, b \in \mathbb{R}, (a + b) - b \in J \Rightarrow a \in J$ .

To prove  $\text{BND}(x)$ , there is only **one choice** for  $a$ :  $a = x$ .

But there are **infinitely many** choices for  $b$ . Why choose  $b = 1$ ?

**Heuristic**: because part of the user goal **matches** the pattern  $a + b$ , namely  $x + 1$ .

## Generating the Inference Rules, Part 2

The trouble with  $\text{BND}(x) \Leftarrow \text{BND}((x + 1) - 1)$

Theorem:  $\forall J \in \mathbb{I}, \forall a, b \in \mathbb{R}, (a + b) - b \in J \Rightarrow a \in J$ .

To prove  $\text{BND}(x)$ , there is only **one choice** for  $a$ :  $a = x$ .

But there are **infinitely many** choices for  $b$ . Why choose  $b = 1$ ?

**Heuristic**: because part of the user goal **matches** the pattern  $a + b$ , namely  $x + 1$ .

Another heuristic: approximate/accurate expressions

Theorem:  $\forall J \in \mathbb{I}, \forall a, b \in \mathbb{R}, (a - b) + b \in J \Rightarrow a \in J$ .

## Generating the Inference Rules, Part 2

The trouble with  $\text{BND}(x) \Leftarrow \text{BND}((x + 1) - 1)$

Theorem:  $\forall J \in \mathbb{I}, \forall a, b \in \mathbb{R}, (a + b) - b \in J \Rightarrow a \in J$ .

To prove  $\text{BND}(x)$ , there is only **one choice** for  $a$ :  $a = x$ .

But there are **infinitely many** choices for  $b$ . Why choose  $b = 1$ ?

**Heuristic:** because part of the user goal **matches** the pattern  $a + b$ , namely  $x + 1$ .

Another heuristic: approximate/accurate expressions

Theorem:  $\forall J \in \mathbb{I}, \forall a, b \in \mathbb{R}, (a - b) + b \in J \Rightarrow a \in J$ .

To choose  $b$ , remove a **rounding** from  $a$  so that  $a - b$  looks like an **absolute error**.

## Generating the Inference Rules, Part 2

The trouble with  $\text{BND}(x) \Leftarrow \text{BND}((x + 1) - 1)$

Theorem:  $\forall J \in \mathbb{I}, \forall a, b \in \mathbb{R}, (a + b) - b \in J \Rightarrow a \in J$ .

To prove  $\text{BND}(x)$ , there is only **one choice** for  $a$ :  $a = x$ .

But there are **infinitely many** choices for  $b$ . Why choose  $b = 1$ ?

**Heuristic**: because part of the user goal **matches** the pattern  $a + b$ , namely  $x + 1$ .

Another heuristic: approximate/accurate expressions

Theorem:  $\forall J \in \mathbb{I}, \forall a, b \in \mathbb{R}, (a - b) + b \in J \Rightarrow a \in J$ .

To choose  $b$ , remove a **rounding** from  $a$  so that  $a - b$  looks like an **absolute error**.

More generally, Gappa stores a set of pairs of **approximate/accurate** expressions.

Here, any expression whose approximate counterpart is  $a$  can be used for  $b$ .

# Beyond Saturation, Proof by Bisection

## Studying cases

If both sets of clauses  $\Gamma \cup \{P\}$  and  $\Gamma \cup \{\neg P\}$  can be refuted, the set  $\Gamma$  is inconsistent.

# Beyond Saturation, Proof by Bisection

## Studying cases

If both sets of clauses  $\Gamma \cup \{P\}$  and  $\Gamma \cup \{\neg P\}$  can be refuted, the set  $\Gamma$  is inconsistent.

Corollary: Assuming  $J_1 \cup \dots \cup J_n = \mathbb{R}$ ,

if all the sets  $\Gamma \cup \{x \in J_k\}$  can be refuted, the set  $\Gamma$  is inconsistent.

# Beyond Saturation, Proof by Bisection

## Studying cases

If both sets of clauses  $\Gamma \cup \{P\}$  and  $\Gamma \cup \{\neg P\}$  can be refuted, the set  $\Gamma$  is inconsistent.

Corollary: Assuming  $J_1 \cup \dots \cup J_n = \mathbb{R}$ ,  
if all the sets  $\Gamma \cup \{\langle x \in J_k \rangle\}$  can be refuted, the set  $\Gamma$  is inconsistent.

## Bisecting intervals

- 1 If not given by the user, infer some expression  $x$  from the goal.
- 2 Once saturation no longer progresses, look for a clause  $\langle x \in J \rangle \in \Gamma$ .
- 3 Create two sets  $\Gamma_k = \Gamma \cup \{\langle x \in J_k \rangle\}$  with  $J_1 \cup J_2 = J$ .
- 4 Saturate both  $\Gamma_k$  until a certain clause or an inconsistency is found.
- 5 If this fails or times out, recursively split  $J_k$  and go back to step 3.

# Summary

- Gappa works by **saturation** and **bisection**.

# Summary

- Gappa works by **saturation** and **bisection**.
- Inference rules are generated once before the saturation.

# Summary

- Gappa works by **saturation** and **bisection**.
- Inference rules are generated once before the saturation.
- Their choice is guided by the **syntax** of the goal, with some heuristics to fill the holes.

# Summary

- Gappa works by **saturation** and **bisection**.
- Inference rules are generated once before the saturation.
- Their choice is guided by the **syntax** of the goal, with some heuristics to fill the holes.
- Users can influence these heuristics and guide the bisections.

# Summary

- Gappa works by **saturation** and **bisection**.
- Inference rules are generated once before the saturation.
- Their choice is guided by the **syntax** of the goal, with some heuristics to fill the holes.
- Users can influence these heuristics and guide the bisections.
- Users can add some **custom rules** by providing equalities.

# Summary

- Gappa works by **saturation** and **bisection**.
  - Inference rules are generated once before the saturation.
  - Their choice is guided by the **syntax** of the goal, with some heuristics to fill the holes.
  - Users can influence these heuristics and guide the bisections.
  - Users can add some **custom rules** by providing equalities.
- 
- The way the goal is written is critical!
  - Gappa will fail to prove something as simple as  $x + 0 = x$ .

# Outline

- 1 Gappa's proof engine
- 2 The database of theorems
  - Forward propagation
  - Binary representation
  - Rounding operators
  - Rewriting rules
  - Summary
- 3 Examples
- 4 Conclusion

# The Database of Theorems

## Requirements

- Intricate reasoning should emerge from the database.
- Theorems should be simple to implement and to formally prove and check.
- The DFS during the first stage should not diverge!

# Interval Arithmetic

## Basic interval arithmetic

- $\text{BND}(a \star b) \Leftarrow \text{BND}(a) \wedge \text{BND}(b)$  for  $\star \in \{+, -, \times, \div, \sqrt{\cdot}\}$ .

# Interval Arithmetic

## Basic interval arithmetic

- $\text{BND}(a \star b) \Leftarrow \text{BND}(a) \wedge \text{BND}(b)$  for  $\star \in \{+, -, \times, \div, \sqrt{\cdot}\}$ .
- $\text{LIN}(a \pm b, c) \Leftarrow \text{LIN}(a, c) \wedge \text{LIN}(b, c)$ .

# Interval Arithmetic

## Basic interval arithmetic

- $\text{BND}(a \star b) \Leftarrow \text{BND}(a) \wedge \text{BND}(b)$  for  $\star \in \{+, -, \times, \div, \sqrt{\cdot}\}$ .
- $\text{LIN}(a \pm b, c) \Leftarrow \text{LIN}(a, c) \wedge \text{LIN}(b, c)$ .

## Fancy theorems

- $\text{ABS}(x + y) \Leftarrow \text{ABS}(x) \wedge \text{ABS}(y)$ ,

# Interval Arithmetic

## Basic interval arithmetic

- $\text{BND}(a \star b) \Leftarrow \text{BND}(a) \wedge \text{BND}(b)$  for  $\star \in \{+, -, \times, \div, \sqrt{\cdot}\}$ .
- $\text{LIN}(a \pm b, c) \Leftarrow \text{LIN}(a, c) \wedge \text{LIN}(b, c)$ .

## Fancy theorems

- $\text{ABS}(x + y) \Leftarrow \text{ABS}(x) \wedge \text{ABS}(y)$ ,
- $\text{REL}(a \cdot b, c \cdot d) \Leftarrow \text{REL}(a, c) \wedge \text{REL}(b, d)$ .

# Binary Representation

## Forward propagation

- $\text{FIX}(a \pm b) \Leftarrow \text{FIX}(a) \wedge \text{FIX}(b)$  and  $\text{FIX}(a \cdot b) \Leftarrow \text{FIX}(a) \wedge \text{FIX}(b)$ ,

# Binary Representation

## Forward propagation

- $\text{FIX}(a \pm b) \Leftarrow \text{FIX}(a) \wedge \text{FIX}(b)$  and  $\text{FIX}(a \cdot b) \Leftarrow \text{FIX}(a) \wedge \text{FIX}(b)$ ,
- $\text{FLT}(a \cdot b) \Leftarrow \text{FLT}(a) \wedge \text{FLT}(b)$ .

# Binary Representation

## Forward propagation

- $\text{FIX}(a \pm b) \Leftarrow \text{FIX}(a) \wedge \text{FIX}(b)$  and  $\text{FIX}(a \cdot b) \Leftarrow \text{FIX}(a) \wedge \text{FIX}(b)$ ,
- $\text{FLT}(a \cdot b) \Leftarrow \text{FLT}(a) \wedge \text{FLT}(b)$ .

## Fancy theorems

- $\text{FLT}(a) \Leftarrow \text{FIX}(a) \wedge \text{ABS}(a)$  and  $\text{FIX}(a) \Leftarrow \text{FLT}(a) \wedge \text{ABS}(a)$ ,

# Binary Representation

## Forward propagation

- $\text{FIX}(a \pm b) \Leftarrow \text{FIX}(a) \wedge \text{FIX}(b)$  and  $\text{FIX}(a \cdot b) \Leftarrow \text{FIX}(a) \wedge \text{FIX}(b)$ ,
- $\text{FLT}(a \cdot b) \Leftarrow \text{FLT}(a) \wedge \text{FLT}(b)$ .

## Fancy theorems

- $\text{FLT}(a) \Leftarrow \text{FIX}(a) \wedge \text{ABS}(a)$  and  $\text{FIX}(a) \Leftarrow \text{FLT}(a) \wedge \text{ABS}(a)$ ,
- $\text{FLT}(a + b) \Leftarrow \text{FLT}(a) \wedge \text{FLT}(b) \wedge \text{LIN}(a, b)$ .

# Rounding Operators

## Forward propagation

- $\text{BND}(\circ(a)) \Leftarrow \text{BND}(a),$

# Rounding Operators

## Forward propagation

- $\text{BND}(\circ(a)) \Leftarrow \text{BND}(a),$
- $\text{FIX}(\circ(a))$  and  $\text{FLT}(\circ(a)),$

# Rounding Operators

## Forward propagation

- $\text{BND}(\circ(a)) \Leftarrow \text{BND}(a)$ ,
- $\text{FIX}(\circ(a))$  and  $\text{FLT}(\circ(a))$ ,
- $\text{FIX}(\circ(a)) \Leftarrow \text{FIX}(a)$  and  $\text{FLT}(\circ(a)) \Leftarrow \text{FLT}(a)$ .

# Rounding Operators

## Forward propagation

- $\text{BND}(\circ(a)) \Leftarrow \text{BND}(a)$ ,
- $\text{FIX}(\circ(a))$  and  $\text{FLT}(\circ(a))$ ,
- $\text{FIX}(\circ(a)) \Leftarrow \text{FIX}(a)$  and  $\text{FLT}(\circ(a)) \Leftarrow \text{FLT}(a)$ .

## Atomic errors

- $\text{BND}(\circ(a) - a) \Leftarrow \text{BND}(a)$ ,

# Rounding Operators

## Forward propagation

- $\text{BND}(\circ(a)) \Leftarrow \text{BND}(a)$ ,
- $\text{FIX}(\circ(a))$  and  $\text{FLT}(\circ(a))$ ,
- $\text{FIX}(\circ(a)) \Leftarrow \text{FIX}(a)$  and  $\text{FLT}(\circ(a)) \Leftarrow \text{FLT}(a)$ .

## Atomic errors

- $\text{BND}(\circ(a) - a) \Leftarrow \text{BND}(a)$ ,
- $\text{REL}(\circ(a), a) \Leftarrow \text{FIX}(a)$ .

# Rounding Operators

## Forward propagation

- $\text{BND}(\circ(a)) \Leftarrow \text{BND}(a)$ ,
- $\text{FIX}(\circ(a))$  and  $\text{FLT}(\circ(a))$ ,
- $\text{FIX}(\circ(a)) \Leftarrow \text{FIX}(a)$  and  $\text{FLT}(\circ(a)) \Leftarrow \text{FLT}(a)$ .

## Atomic errors

- $\text{BND}(\circ(a) - a) \Leftarrow \text{BND}(a)$ ,
- $\text{REL}(\circ(a), a) \Leftarrow \text{FIX}(a)$ .

## Formats

- $\text{EQL}(\circ(a), a) \Leftarrow \text{FIX}(a) \wedge \text{FLT}(a)$ .

# Rewriting Rules

## Error analysis

- $\text{BND}(a \cdot b - c \cdot d) \Leftarrow \text{BND}((a - c) \cdot d + c \cdot (b - d) + (a - c) \cdot (b - d)),$

# Rewriting Rules

## Error analysis

- $\text{BND}(a \cdot b - c \cdot d) \Leftarrow \text{BND}((a - c) \cdot d + c \cdot (b - d) + (a - c) \cdot (b - d)),$
- $\text{LIN}(a - b, c) \Leftarrow \text{LIN}((a - d) + (d - b), c),$

# Rewriting Rules

## Error analysis

- $\text{BND}(a \cdot b - c \cdot d) \Leftarrow \text{BND}((a - c) \cdot d + c \cdot (b - d) + (a - c) \cdot (b - d)),$
- $\text{LIN}(a - b, c) \Leftarrow \text{LIN}((a - d) + (d - b), c),$
- $\text{REL}(a \cdot b, a \cdot c) \Leftarrow \text{REL}(b, c),$

# Rewriting Rules

## Error analysis

- $\text{BND}(a \cdot b - c \cdot d) \Leftarrow \text{BND}((a - c) \cdot d + c \cdot (b - d) + (a - c) \cdot (b - d)),$
- $\text{LIN}(a - b, c) \Leftarrow \text{LIN}((a - d) + (d - b), c),$
- $\text{REL}(a \cdot b, a \cdot c) \Leftarrow \text{REL}(b, c),$
- $\text{REL}(a + b, c + d) \Leftarrow \text{REL}(a, c) \wedge \text{REL}(b, d) \wedge \text{LIN}(c, c + d),$

# Rewriting Rules

## Error analysis

- $\text{BND}(a \cdot b - c \cdot d) \Leftarrow \text{BND}((a - c) \cdot d + c \cdot (b - d) + (a - c) \cdot (b - d)),$
- $\text{LIN}(a - b, c) \Leftarrow \text{LIN}((a - d) + (d - b), c),$
- $\text{REL}(a \cdot b, a \cdot c) \Leftarrow \text{REL}(b, c),$
- $\text{REL}(a + b, c + d) \Leftarrow \text{REL}(a, c) \wedge \text{REL}(b, d) \wedge \text{LIN}(c, c + d),$
- $\text{BND}(\sqrt{a} - \sqrt{b}) \Leftarrow \text{BND}((a - b)/(\sqrt{a} + \sqrt{b})) \wedge \dots$

# Rewriting Rules

## Error analysis

- $\text{BND}(a \cdot b - c \cdot d) \Leftarrow \text{BND}((a - c) \cdot d + c \cdot (b - d) + (a - c) \cdot (b - d)),$
- $\text{LIN}(a - b, c) \Leftarrow \text{LIN}((a - d) + (d - b), c),$
- $\text{REL}(a \cdot b, a \cdot c) \Leftarrow \text{REL}(b, c),$
- $\text{REL}(a + b, c + d) \Leftarrow \text{REL}(a, c) \wedge \text{REL}(b, d) \wedge \text{LIN}(c, c + d),$
- $\text{BND}(\sqrt{a} - \sqrt{b}) \Leftarrow \text{BND}((a - b)/(\sqrt{a} + \sqrt{b})) \wedge \dots$

## Backward propagation

- $\text{BND}(a) \Leftarrow \text{BND}((a \cdot b)/b) \wedge \text{NZR}(b).$

# Summary

- Most of theorems are based on **naive interval** arithmetic, which makes them easy to implement and to check a posteriori.

# Summary

- Most of theorems are based on **naive interval** arithmetic, which makes them easy to implement and to check a posteriori.
- The LIN and REL predicates are just a way to avoid divisions by zero. The ABS predicate is just a way to avoid doing sign-based bisections.

# Summary

- Most of theorems are based on **naive interval** arithmetic, which makes them easy to implement and to check a posteriori.
- The LIN and REL predicates are just a way to avoid divisions by zero. The ABS predicate is just a way to avoid doing sign-based bisections.
- Part of the domain-specific knowledge goes into the FIX and FLT predicates.

# Summary

- Most of theorems are based on **naive interval** arithmetic, which makes them easy to implement and to check a posteriori.
- The LIN and REL predicates are just a way to avoid divisions by zero. The ABS predicate is just a way to avoid doing sign-based bisections.
- Part of the domain-specific knowledge goes into the FIX and FLT predicates.
- Theorems based on “rewriting” simulate the way we usually reason about **rounding errors** at first order.

# Summary

- Most of theorems are based on **naive interval** arithmetic, which makes them easy to implement and to check a posteriori.
  - The LIN and REL predicates are just a way to avoid divisions by zero. The ABS predicate is just a way to avoid doing sign-based bisections.
  - Part of the domain-specific knowledge goes into the FIX and FLT predicates.
  - Theorems based on “rewriting” simulate the way we usually reason about **rounding errors** at first order.
- 
- Handling error cancellations requires some help from the user.
  - If you know something (e.g., exact operations), do not hide it from the tool.

# Outline

- 1 Gappa's proof engine
- 2 The database of theorems
- 3 Examples
  - Fast nearbyint
  - Newton-Raphson division
- 4 Conclusion

# Fast Nearbyint

In binary64,  $y = \circ(\circ(x + 3 \cdot 2^{51}) - 3 \cdot 2^{51})$  is an efficient way to compute the closest integer from  $x$  when  $|x| \leq 2^{51}$ .

How to verify it with Gappa?

# Fast Nearbyint

In binary64,  $y = \circ(\circ(x + 3 \cdot 2^{51}) - 3 \cdot 2^{51})$  is an efficient way to compute the closest integer from  $x$  when  $|x| \leq 2^{51}$ .

How to verify it with Gappa?

## Stating the problem

# First, a few definitions to improve readability:

```
@rnd = float<ieee_64,ne>;    # shorter name for the rounding operator
x = rnd(x_);                  # simple way to state that x is actually a FP number
y rnd= (x + 3b51) - 3b51;     # same as y = rnd(rnd(x + 3b51) - 3b51)
```

# Then, the formula we want to prove:

```
{ x in [-1b51,1b51] -> @FIX(y,0) /\ |y - x| <= 0.5 }
```

# Fast Nearbyint

In binary64,  $y = \circ(\circ(x + 3 \cdot 2^{51}) - 3 \cdot 2^{51})$  is an efficient way to compute the closest integer from  $x$  when  $|x| \leq 2^{51}$ .

How to verify it with Gappa?

## Results

- Gappa successfully proves that  $y$  is a multiple of  $2^0$ .
- Gappa only proves that  $|y - x| \leq 2^{52}$  (naive enclosure).

# Fast Nearbyint

In binary64,  $y = \circ(\circ(x + 3 \cdot 2^{51}) - 3 \cdot 2^{51})$  is an efficient way to compute the closest integer from  $x$  when  $|x| \leq 2^{51}$ .

How to verify it with Gappa?

## Results

- Gappa successfully proves that  $y$  is a multiple of  $2^0$ .
- Gappa only proves that  $|y - x| \leq 2^{52}$  (naive enclosure).

## Cause of the failure

- $y = \circ(\circ(x + 3 \cdot 2^{51}))$  and  $x$  have different syntactic structures, so no theorem for error analysis can be applied.
- But  $y$  and  $(x + 3 \cdot 2^{51}) - 3 \cdot 2^{51}$  would have the same structure.

# Fast Nearbyint

In binary64,  $y = \circ(\circ(x + 3 \cdot 2^{51}) - 3 \cdot 2^{51})$  is an efficient way to compute the closest integer from  $x$  when  $|x| \leq 2^{51}$ .

How to verify it with Gappa?

## Full script

```
# First, a few definitions to improve readability:
@rnd = float<ieee_64,ne>;    # shorter name for the rounding operator
x = rnd(x_);                  # simple way to state that x is actually a FP number
y rnd= (x + 3b51) - 3b51;     # same as y = rnd(rnd(x + 3b51) - 3b51)

# Then, the formula we want to prove:
{ x in [-1b51,1b51] -> @FIX(y,0) /\ |y - x| <= 0.5 }

# Finally, an axiom to guide the tool:
(x + 3b51) - 3b51 -> x;
```

# Newton-Raphson Division

## Itanium 16-bit unsigned integer division $\lfloor a/b \rfloor$

```
@rnd = float<x86_80,ne>;
y0 = frcpa(a, b);
q0 = rnd(a * y0);
e0 = rnd(1 + 1b-17 - b * y0);    # FMA
q1 = rnd(q0 + e0 * q0);          # FMA
result = int<dn>(q1);
```

```
{ @FIX(a, 0) /\ a in [1,65535] ->                # the inputs a and b are 16-bit integers
  @FIX(b, 0) /\ b in [1,65535] ->
  result = int<dn>(a / b) }
```

# Newton-Raphson Division

## Itanium 16-bit unsigned integer division $\lfloor a/b \rfloor$

```
@rnd = float<x86_80,ne>;
y0 = frcpa(a, b);
q0 = rnd(a * y0);
e0 = rnd(1 + 1b-17 - b * y0);    # FMA
q1 = rnd(q0 + e0 * q0);          # FMA
result = int<dn>(q1);

{ @FIX(a, 0) /\ a in [1,65535] ->                # the inputs a and b are 16-bit integers
  @FIX(b, 0) /\ b in [1,65535] ->
  result = int<dn>(a / b) }
```

- Gappa does not know frcpa, so let us use its specification instead.
- The lemma  $0 \leq bq - a < 1 \Rightarrow \lfloor q \rfloor = \lfloor a/b \rfloor$  is out of Gappa's scope.

# Newton-Raphson Division

## Itanium 16-bit unsigned integer division $\lfloor a/b \rfloor$

```

@rnd = float<x86_80,ne>;
# computed values          # exact values
q0 = rnd(a * y0);          Mq0 = a * y0;
e0 = rnd(1 + 1b-17 - b * y0); Me0 = 1 + 1b-17 - b * y0;
q1 = rnd(q0 + e0 * q0);    Mq1 = Mq0 + Me0 * Mq0;

{ @FIX(a, 0) /\ a in [1,65535] ->          # the inputs a and b are 16-bit integers
  @FIX(b, 0) /\ b in [1,65535] ->
  @FLT(y0, 11) /\ |(y0 - 1/b) / (1/b)| <= 0.00211373 ->      # specification of frcpa
  a * ((q1 - a/b) / (a/b)) in [0,0.9999] }

```

# Newton-Raphson Division

## Itanium 16-bit unsigned integer division $\lfloor a/b \rfloor$

```
@rnd = float<x86_80,ne>;
# computed values          # exact values
q0 = rnd(a * y0);          Mq0 = a * y0;
e0 = rnd(1 + 1b-17 - b * y0); Me0 = 1 + 1b-17 - b * y0;
q1 = rnd(q0 + e0 * q0);    Mq1 = Mq0 + Me0 * Mq0;

{ @FIX(a, 0) /\ a in [1,65535] ->          # the inputs a and b are 16-bit integers
  @FIX(b, 0) /\ b in [1,65535] ->
  @FLT(y0, 11) /\ |(y0 - 1/b) / (1/b)| <= 0.00211373 ->      # specification of frcpa
  a * ((q1 - a/b) / (a/b)) in [0,0.9999] }
```

- Gappa does not know about the quadratic convergence of Newton-Raphson. And because of  $2^{-17}$ , it is not even a Newton-Raphson iteration anyway.

# Newton-Raphson Division

## Itanium 16-bit unsigned integer division $\lfloor a/b \rfloor$

```

@rnd = float<x86_80,ne>;
# computed values          # exact values
q0 = rnd(a * y0);          Mq0 = a * y0;
e0 = rnd(1 + 1b-17 - b * y0); Me0 = 1 + 1b-17 - b * y0;
q1 = rnd(q0 + e0 * q0);    Mq1 = Mq0 + Me0 * Mq0;

eps0 = (y0 - 1/b) / (1/b);

{ @FIX(a, 0) /\ a in [1,65535] ->          # the inputs a and b are 16-bit integers
  @FIX(b, 0) /\ b in [1,65535] ->
  @FLT(y0, 11) /\ |eps0| <= 0.00211373 -> # specification of frcpa
  a * ((q1 - a/b) / (a/b)) in [0,0.9999] }

# not quite quadratic convergence:
(Mq1 - a/b) / (a/b) -> -(eps0 * eps0) + (1 + eps0) * 1b-17 { a <> 0, b <> 0 };

```

# Newton-Raphson Division

## Itanium 16-bit unsigned integer division $\lfloor a/b \rfloor$ , the Rocq version

```

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

```

Proof.

```

...      (* 40 lines to prove the quadratic convergence and the b*q-a cut *)
gappa.   (* a 24kB generated proof, 5kB compressed *)

```

Qed.

# Outline

- 1 Gappa's proof engine
- 2 The database of theorems
- 3 Examples
- 4 Conclusion

# Some Other Approaches

## Floating-point theory of SMT-LIB

- **Standardized** format (CVC5, Z3, etc).
- Handle **exceptional cases**.
- **Complete** and sometimes efficient.
- **Errors are out of scope**.

# Some Other Approaches

## Floating-point theory of SMT-LIB

- **Standardized** format (CVC5, Z3, etc).
- Handle **exceptional cases**.
- **Complete** and sometimes efficient.
- **Errors are out of scope**.

## FPTaylor

- **First-order symbolic** expansions.
- Limited to the **standard model**.
- Proofs for the **HOL Light** system.

# Some Other Approaches

## Floating-point theory of SMT-LIB

- **Standardized** format (CVC5, Z3, etc).
- Handle **exceptional cases**.
- **Complete** and sometimes efficient.
- **Errors are out of scope**.

## FPTaylor

- **First-order symbolic** expansions.
- Limited to the **standard model**.
- Proofs for the **HOL Light** system.

## Zhang & Aiken's SELTZO

- **Specialized** for FP **expansions**.
- **Impossible to audit** proofs.
- Limited to **power-of-two error** bounds.

# Conclusion

## Gappa

- Support for binary fixed- and floating-point arithmetic.
- Generation of formal proofs for the Rocq system; also available as a tactic.
- Somewhat readable LaTeX and D2 proofs.
- Fast and usable in an “interactive” way.

## Shortcomings

- Only useful for short sequences of operations.
- Heavily reliant on user annotations and hence expertise.
- Poor support for counterexamples.

<https://gappa.gitlabpages.inria.fr/>