Journées Arinews

Formal specifications of floating-point programs

Sylvie Boldo

INRIA Futurs, LRI

November 2005

Disclaimer 1

This is work in progress!

Any idea or suggestion is welcomed.

Outline

Motivations

How to guarantee a critical program?

Several (complementary) approaches:

- tests (exhaustiveness?),
- pen & paper proof (safety?),
- formal proof.

How to guarantee a critical program?

Several (complementary) approaches:

- tests (exhaustiveness?),
- pen & paper proof (safety?),
- formal proof.

Formal methods really increase the trust the user may have in the algorithm.

How to guarantee a critical program?

Several (complementary) approaches:

- tests (exhaustiveness?),
- pen & paper proof (safety?),
- formal proof.

Formal methods really increase the trust the user may have in the algorithm.

However, the proof and the formal stating of the theorem are hard to read and hard to understand by non-specialists.

How to guarantee a critical program?

How to be sure of this equality?

the proved "stuff" = the "stuff" in my plane

How to guarantee a critical program?

How to be sure of this equality?

the proved "stuff" = the "stuff" in my plane

You have to prove the program as it is written and make the formal statement understandable.

Existing tools

- Proof assistants -

Coq (http://coq.inria.fr)

Coq is a formal proof checker, meaning it only checks a proof (it also generates easy demonstrations).

Coq (http://coq.inria.fr)

Coq is a formal proof checker, meaning it only checks a proof (it also generates easy demonstrations).

- few automations (automatic proofs, powerful tactics)
- Coq kernel mechanically checks each step of the proof
- in use, you successively apply tactics (lemma invocation, rewriting, simplification, hypotheses management...) to reduce the goal down to the hypotheses.

Other proof assistants

There are other inhabitants in formal methods' Wonderland:

Other proof assistants

There are other inhabitants in formal methods' Wonderland:

- Isabelle/HOL Light (the sister)

Other proof assistants

There are other inhabitants in formal methods' Wonderland:

- Isabelle/HOL Light (the sister)
- PVS (the cousin, whose ways are questionable, yet he is successful)

Other proof assistants

There are other inhabitants in formal methods' Wonderland:

- Isabelle/HOL Light (the sister)
- PVS (the cousin, whose ways are questionable, yet he is successful)
- ACL2 (the grand-father, a little old-fashioned, yet still fit)

Other proof assistants

There are other inhabitants in formal methods' Wonderland:

- Isabelle/HOL Light (the sister)
- PVS (the cousin, whose ways are questionable, yet he is successful)
- ACL2 (the grand-father, a little old-fashioned, yet still fit)
- Simplify (the pet: not bright, but much brute force)

Other proof assistants

There are other inhabitants in formal methods' Wonderland:

- Isabelle/HOL Light (the sister)
- PVS (the cousin, whose ways are questionable, yet he is successful)
- ACL2 (the grand-father, a little old-fashioned, yet still fit)
- Simplify (the pet: not bright, but much brute force)

  (It is called a decision procedure.)

Existing tools

- Formalizations of FP arithmetic -

Formalizations

Existence:

- Formalization Coq since 2001 (AOC, Daumas-Rideau-Théry).
- Formalization PVS (same one) since 2005 (Boldo, Muñoz).

Formalizations

Existence:

- Formalization Coq since 2001 (AOC, Daumas-Rideau-Théry).
- Formalization PVS (same one) since 2005 (Boldo, Muñoz).

Characteristics:

- Normal floats, subnormal floats, cohorts, generic rounding, IEEE roundings, no Overflow.
- Many floats may share the same real value.
- High level formalization with mathematical definitions.

Formalization

Float = pair of signed integers (significand, exponent)

$$(n, e) \in \mathbb{Z}^2$$

Formalization

Float = pair of signed integers (significand, exponent)
associated to a real value.

$$(n, e) \in \mathbb{Z}^2 \quad \hookrightarrow \quad n \times \beta^e \in \mathbb{R}$$

Formalization

Float = pair of signed integers (significand, exponent)
associated to a real value.

$$(n, e) \in \mathbb{Z}^2 \quad \hookrightarrow \quad n \times \beta^e \in \mathbb{R}$$

1.00010_2 E 4 $\mapsto$ $(100010_2, -1)_2$ $\hookrightarrow$ 17

IEEE-754 significand of the 754R real value

Existing tools

- Why -

Method

The method is to annotate the ML/C/Java program.

Method

The method is to annotate the ML/C/Java program.

We add as comments pre-conditions and post-conditions to functions (and which variables are modified).

Method

The method is to annotate the ML/C/Java program.

We add as comments pre-conditions and post-conditions to functions (and which variables are modified).

We tell the variants and invariants of the loops.

We add assertions.

...

Method

The method is to annotate the ML/C/Java program.

We add as comments pre-conditions and post-conditions to functions (and which variables are modified).

We tell the variants and invariants of the loops.

We add assertions.

...

The tool generates proof obligations associated to the user annotations.

There is only these verification conditions to prove.

Why

Written by Jean-Christophe Filliâtre.

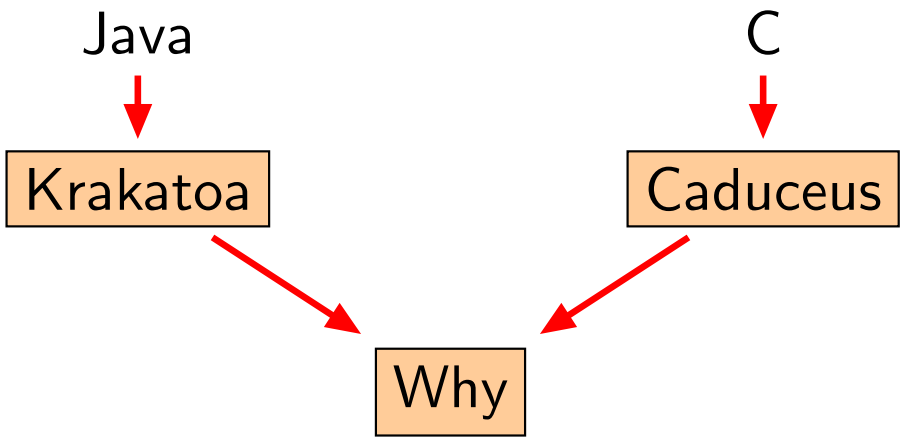
Java C

Why

Written by Jean-Christophe Filliâtre.

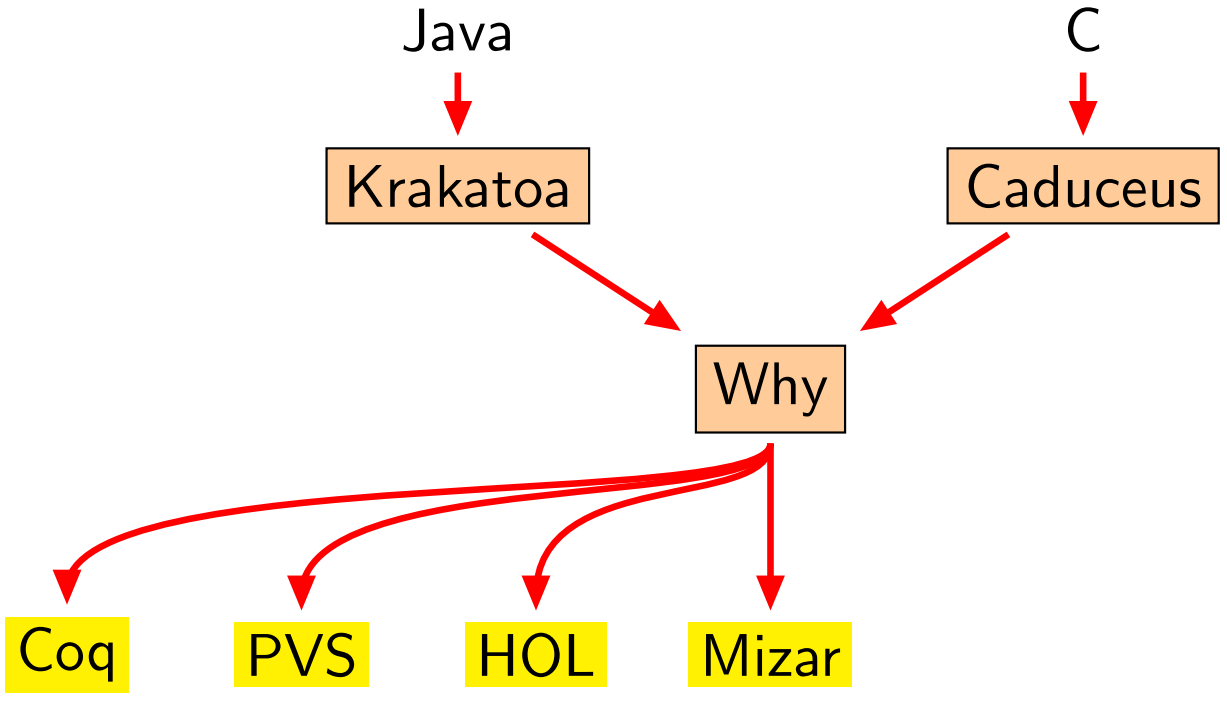
Why

Written by Jean-Christophe Filliâtre.



Why

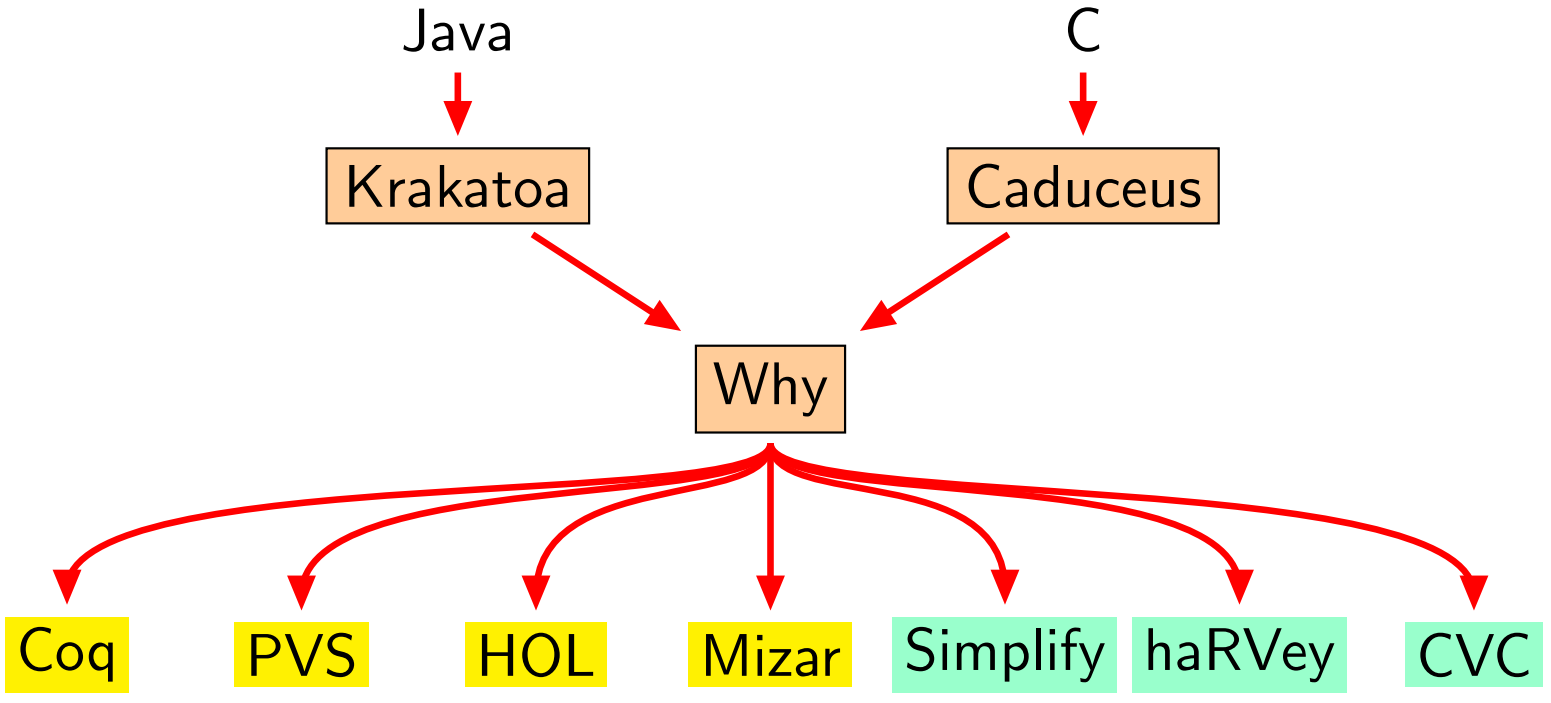
Written by Jean-Christophe Filliâtre.



Verification conditions

Why

Written by Jean-Christophe Filliâtre.



Verification conditions

Example 1

```
let swap =
  fun (t:int array)(i,j:int) ->
    { 0 <= i < array_length(t)
        and 0 <= j < array_length(t) }
    (let v = t[i] in
    begin
     t[i] := t[j];
     t[j] := v
    end)
    { exchange(t, t@, i, j) }
```

where exchange is a proof-assistant defined function.

Example 2

```
let rec quick_rec (t:int array)(l,r:int) :  unit
 { variant 1+r-l } =
  { 0 <= l and r < array_length(t) }
  (if l < r then
   let p = (partition t l r) in
   begin
    (quick_rec t l (p-1));
    (quick_rec t (p+1) r)
   end)
{ sorted_array(t, l, r) and sub_permut(l, r, t, t@) }
```

Consequences

Now, `float` and `double` are interpreted as real numbers by Why!

Consequences

Now, `float` and `double` are interpreted as real numbers by Why!

We want to combine Why with our formalizations to be able to prove floating-point programs.

Consequences

Now, `float` and `double` are interpreted as real numbers by Why!

We want to combine Why with our formalizations to be able to prove floating-point programs.

To do that, we must define the annotations we will use and the model for floating-point arithmetic.

Specification language

Correct result

What is often wanted to express is that “the obtained result is near the correct result”. But what is the correct result?

Correct result

What is often wanted to express is that "the obtained result is near the correct result". But what is the correct result?

- a known value, such as π,

Correct result

What is often wanted to express is that "the obtained result is near the correct result". But what is the correct result?

- a known value, such as π,
- a mathematical value depending on the inputs, such as $\cos(x)$,

Correct result

What is often wanted to express is that "the obtained result is near the correct result". But what is the correct result?

- a known value, such as π,
- a mathematical value depending on the inputs, such as $\cos(x)$,
- the result we would have had if no FP errors had occurred.

Correct result

What is often wanted to express is that "the obtained result is near the correct result". But what is the correct result?

- a known value, such as π,
- a mathematical value depending on the inputs, such as $\cos(x)$,
- the result we would have had if no FP errors had occurred.

Ouch...

Correct result

What is the correct result of Malcolm-Gentleman's algorithm?

```
A=2;
while (A != (A+1))
  A*=2;
```

Correct result

What is the correct result of Malcolm-Gentleman's algorithm?

```
A=2;
while (A != (A+1))
  A*=2;
```

If there is no FP error: $+\infty$.

If all the computations are correct, and if we follow the same path as for the true computation: 2^p (using radix 2).

Correct result

What is the correct result of Malcolm-Gentleman's algorithm?

```
A=2;
while (A != (A+1))
  A*=2;
```

If there is no FP error: $+\infty$.

If all the computations are correct, and if we follow the same path as for the true computation: 2^p (using radix 2).

We will compare the FP result to the result obtained if all the computations are exact, but all the tests are made on the FP data.

Correct result

What is the correct result of Malcolm-Gentleman's algorithm?

```
A=2;
while (A != (A+1))
  A*=2;
```

If there is no FP error: $+\infty$.

If all the computations are correct, and if we follow the same path as for the true computation: 2^p (using radix 2).

We will compare the FP result to the result obtained if all the computations are exact, but all the tests are made on the FP data.

The user may add assertions telling the path chosen is the same in both cases.

Disclaimer 2

The following assertions are not proved.

the following examples are not parsed by Why.

Example 1

```
/*@ requires y/2 <= x <= 2*y
  @ ensures \err = 0 */

double Sterbenz(double x, double y) {
   return x-y; }
```

Example 2

```
/*@ requires x in [0,1]
  @ ensures \result in [161,276]
  @   and \err(\result) <= 2*ulp(\result) */

double calcul(double x) {
   double a,b;
   a=2*x+3;
   b=x*x+55;
   a=a*b-4;
   return a; }
```

Example 3

```
/*@ ensures \result = exp2(53) */

  double malcolm1() {
   double A;
   A=2;   /*@ assert A=2 */

   /*@ invariant 0 <= A and A <= exp2(53)
     @ variant exp2(53)-A */
   while (A != (A+1)) {
     A*=2;
     /*@ assert A = 2*A@ */
   }
   return A; }
```

Conclusion

Conclusion 1

Inside Why, we will keep both the computed float and the correct result (of our definition).

Conclusion 1

Inside Why, we will keep both the computed float and the correct result (of our definition).

$\Rightarrow$ We may use `\err(x)` in the annotations.

Conclusion 1

Inside Why, we will keep both the computed float and the correct result (of our definition).

$\Rightarrow$ We may use `\err(x)` in the annotations.

$\Rightarrow$ The Why float is more than an IEEE float.

Conclusion 1

Inside Why, we will keep both the computed float and the correct result (of our definition).

$\Rightarrow$ We may use `\err(x)` in the annotations.

$\Rightarrow$ The Why float is more than an IEEE float.

$\Rightarrow$ `assert A=2` does not have a clear meaning.

$\Rightarrow$ `assert A=2` is a typing problem.

Conclusion 2

We want to provide the user with useful floating-point functions.

- `\err`,
- `ulp` (which one?),
- `exp2`.

What is missing?

Conclusion 2

We want to provide the user with useful floating-point functions.

- `\err`,
- `ulp` (which one?),
- `exp2`.

What is missing?

Note that the user may add his own functions if he defines them in the proof assistant.

Conclusion 3

What are the possible user modes?

Mode	radix	precision	
normal	2	53	
intel	2	64	for all computations?
generic-I	2	p	
generic-II	β	p	do we require $\beta = 2$ or 10?
			do we require β even?

Thank you for your attention!