An Automatable Formal Semantics for IEEE-754 Floating-Point Arithmetic

Martin Brain, Cesare Tinelli, Philipp Rümmer, Thomas Wahl
(and the rest of the SMT community)

University of Oxford

June 24, 2015

Hasn't this been done before?

Isabelle	*A formal model of IEEE oating point arithmetic*
HOL	*Interpretation of IEEE-854 oating-point standard and de nition in the HOL system.*
HOL Light	*Floating point veri cation in HOL light: The exponential function* (Intel)
ACL2	*A mechanically checked proof of the AMD5K86TM oating-point division program* (AMD and Centaur)
PVS	*De ning the IEEE-854 oating-point standard in PVS*
Coq	*A generic library for oating-point numbers and its application to exact computing*
Coq	*Floating-point arithmetic in the Coq system*
Coq	*Flocq: A Uni ed Library for Proving Floating-point Algorithms in Coq*

Hasn't this been done before?

Isabelle	*A formal model of IEEE oating point arithmetic*
HOL	*Interpretation of IEEE-854 oating-point standard and de nition in the HOL system.*
HOL Light	*Floating point veri cation in HOL light: The exponential function* (Intel)
ACL2	*A mechanically checked proof of the AMD5K86TM oating-point division program* (AMD and Centaur)
PVS	*De ning the IEEE-854 oating-point standard in PVS*
Coq	*A generic library for oating-point numbers and its application to exact computing*
Coq	*Floating-point arithmetic in the Coq system*
Coq	*Flocq: A Uni ed Library for Proving Floating-point Algorithms in Coq*

... there is another way to think about theorem-proving ...

Is there an x and y such that ...

$$\begin{aligned} 0 &< x \\ 0 &< y \\ x + y &< x \end{aligned}$$

Is there an x and y such that ...

$$0 \clubsuit x$$

$$0 \clubsuit y$$

$$x \spadesuit y \clubsuit x$$

Is there an x and y such that ...

$$\begin{aligned} 0 \ \clubsuit \ x \\ 0 \ \clubsuit \ y \\ x \spadesuit y \ \clubsuit \ x \end{aligned}$$

It depends on the interpretation (of $\clubsuit$ and $\spadesuit$)!

$$\begin{aligned} D &= \mathbb{Z} \\ [\![\clubsuit]\!] &= <_{\mathbb{Z}} \\ [\![\spadesuit]\!] &= +_{\mathbb{Z}} \end{aligned}$$

NO!

Is there an x and y such that ...

$$\begin{array}{rcl} 0 & \clubsuit & x \\ 0 & \clubsuit & y \\ x \spadesuit y & \clubsuit & x \end{array}$$

It depends on the interpretation (of $\clubsuit$ and $\spadesuit$)!

$$\begin{array}{rcl} D & = & \{00, 01, 10, 11\} \\ [\![\clubsuit]\!] & = & \texttt{bvult} \\ [\![\spadesuit]\!] & = & \texttt{bvplus} \end{array}$$

Yes ($x = 01$, $y = 11$)

First Order Logic

Syntax

Fix a *signature* Σ
(i.e. $\Sigma = \{\clubsuit, \spadesuit\}$)

Semantics

An *interpretation* is
$M = (D, [\![.]\!] : \Sigma \rightarrow (2^{D^n}))$

Satisfiability

An interpretation M *satis es* a formula ϕ:

$$M \models \phi$$

If ϕ evaluated over D (using $[\![.]\!]$) is true.

How Do We Fix The *Meaning* of Symbols?

Option 1 – Axiomatic

$$M \models Axioms \Rightarrow \phi$$

$$\begin{aligned} Axioms &= \forall a, b, c \,.\, a \clubsuit b \wedge b \clubsuit c \Rightarrow a \clubsuit c \\ &= \forall a \,.\, \neg a \clubsuit a \\ &= \ldots \end{aligned}$$

Formalisation is solver *INPUT*.

Pros

+ Easy to implement
+ Flexible
+ Can add theorems

Cons

- All formulae quantified
- Axioms not always simple
- Hard to solve

How Do We Fix The *Meaning* of Symbols?

Option 2 – Algebraic

Fix signature Σ' and its interpretation $M' = (D, [\![.]\!] : \Sigma' \to (2^{D^n}))$.

$$D = \mathbb{Z} \quad [\![\clubsuit]\!] = <_{\mathbb{Z}} \quad [\![\spadesuit]\!] = +_{\mathbb{Z}}$$

Is there M extension of M' such that:

$$M \models \phi$$

Formalisation is solver *SPECIFICATION*.

Pros

+ Fast decision procedures
+ Counter-examples
+ Few quantifiers

Cons

- Theory has to be built into solver
- Implementation harder

How Do We Fix The *Meaning* of Symbols?

Option 2 – Algebraic

Fix signature Σ' and its interpretation $M' = (D, [\![.]\!] : \Sigma' \to (2^{D^n}))$.

$$D = \mathbb{Z} \quad [\![\clubsuit]\!] = <_{\mathbb{Z}} \quad [\![\spadesuit]\!] = +_{\mathbb{Z}}$$

Is there M extension of M' such that:

$$M \models \phi$$

Formalisation is solver *SPECIFICATION*.

Pros

+ Fast decision procedures
+ Counter-examples
+ Few quantifiers

Cons

- Theory has to be built into solver
- Implementation harder

Requirements

Principles

Bit-Exact	Must do *exactly* what the hardware does
Precise	Gives SAT / UNSAT (ideally with model / proof)
Automated	Ideally fast and "out of the box"
Flexible	Support different decision procedures

Target Applications

- Path feasibility / test-case generation
- Generation of special values
- Numerical instability
- Undefined behaviour
- Hardware verification
- Functional correctness
- Automated numerical analysis

IEEE-754 2008

IEEE Std 754-2008
IEEE Standard for Floating-Point Arithmetic

3.2 Specification levels

Floating-point arithmetic is a systematic approximation of real arithmetic, as illustrated in Table 3.1. Floating-point arithmetic can only represent a finite subset of the continuum of real numbers. Consequently certain properties of real arithmetic, such as associativity of addition, do not always hold for floating-point arithmetic.

Table 3.1—Relationships between different specification levels for a particular format

Level 1	$\{-\infty \ldots 0 \ldots +\infty\}$	Extended real numbers.
many-to-one ↓	*rounding*	↑ projection (except for NaN)
Level 2	$\{-\infty \ldots -0\} \cup \{+0 \ldots +\infty\} \cup$ **NaN**	Floating-point data—an algebraically closed system.
one-to-many ↓	*representation specification*	↑ many-to-one
Level 3	(***sign***, ***exponent***, ***significand***) ∪ $\{-\infty, +\infty\}$ ∪ **qNaN** ∪ **sNaN**	Representations of floating-point data.
one-to-many ↓	*encoding for representations of floating-point data*	↑ many-to-one
Level 4	0111000...	Bit strings.

, the set The mathematical structure underpinning the arithmetic in this standard is the extended reals, that is, unding of real numbers together with positive and negative infinity. For a given format, the process of *ro ing-point* (see 4) maps an extended real number to a *floating-point number* included in that format. A *float* mber), can *datum*, which can be a signed zero, finite non-zero number, signed infinity, or a NaN (not-a-num be mapped to one or more *representations of floating-point data* in a format.

Level 1 : Extended Reals

$$\mathbb{R}^* = \mathbb{R} \cup \{+\infty, -\infty, \mathrm{NaN}\}$$

(partially ordered, additive and multiplicative commutative monoid with the distributivity property)

$$\begin{aligned}
u + \mathrm{NaN} = \mathrm{NaN} + u &= \mathrm{NaN} \\
-\mathrm{NaN} &= \mathrm{NaN} \\
u \cdot \mathrm{NaN} = \mathrm{NaN} \cdot u &= \mathrm{NaN} \\
\mathrm{NaN}^{-1} &= \mathrm{NaN} \\
\mathrm{NaN} \leqslant u &\Leftrightarrow u = \mathrm{NaN} \\
u \leqslant \mathrm{NaN} &\Leftrightarrow u = \mathrm{NaN}
\end{aligned}$$

$$\begin{aligned}
+\infty \leqslant w &\Leftrightarrow w = +\infty \\
w \leqslant -\infty &\Leftrightarrow w = -\infty \\
-(+\infty) &= -\infty \\
-(-\infty) &= +\infty \\
+\infty^{-1} &= 0 \\
-\infty^{-1} &= 0 \\
0^{-1} &= +\infty \\
&\dots
\end{aligned}$$

Level 2(ish) : Domain

$$
\begin{aligned}
\mathbb{F}_{\varepsilon,\sigma} &= \mathrm{F}_{\varepsilon,\sigma} \cup \{\mathrm{NaN}\} \\
\mathrm{F}_{\varepsilon,\sigma} &= \mathrm{FZ}_{\varepsilon,\sigma} \cup \mathrm{FS}_{\varepsilon,\sigma} \cup \mathrm{FN}_{\varepsilon,\sigma} \cup \mathrm{FI}_{\varepsilon,\sigma} \\
\mathrm{FZ}_{\varepsilon,\sigma} &= \{(s,e,m) \in B_{\varepsilon,\sigma} \mid e = \mathbf{0}_{\varepsilon},\ m = \mathbf{0}_{\sigma-1}\} \\
\mathrm{FS}_{\varepsilon,\sigma} &= \{(s,e,m) \in B_{\varepsilon,\sigma} \mid e = \mathbf{0}_{\varepsilon},\ m \neq \mathbf{0}_{\sigma-1}\} \\
\mathrm{FN}_{\varepsilon,\sigma} &= \{(s,e,m) \in B_{\varepsilon,\sigma} \mid e \neq \mathbf{1}_{\varepsilon},\ e \neq \mathbf{0}_{\varepsilon}\} \\
\mathrm{FI}_{\varepsilon,\sigma} &= \{(s,e,m) \in B_{\varepsilon,\sigma} \mid e = \mathbf{1}_{\varepsilon},\ m = \mathbf{0}_{\sigma-1}\}
\end{aligned}
$$

$$
\mathrm{v}_{\varepsilon,\sigma} : \mathbb{F}_{\varepsilon,\sigma} \to \mathbb{R}^*
$$

IEEE-754 2008 again

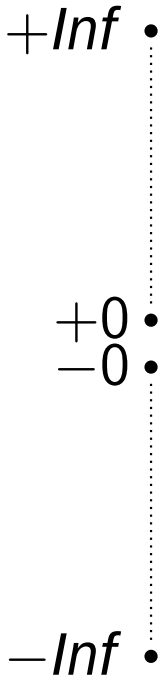
IEEE Std 754-2008
IEEE Standard for Floating-Point Arithmetic

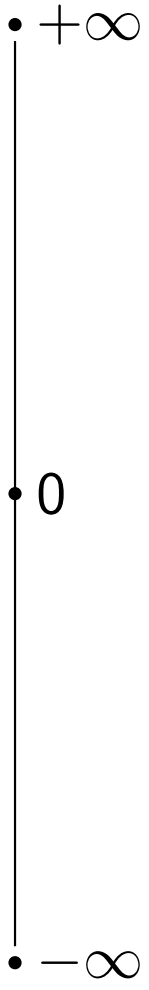
5. Operations

5.1 Overview

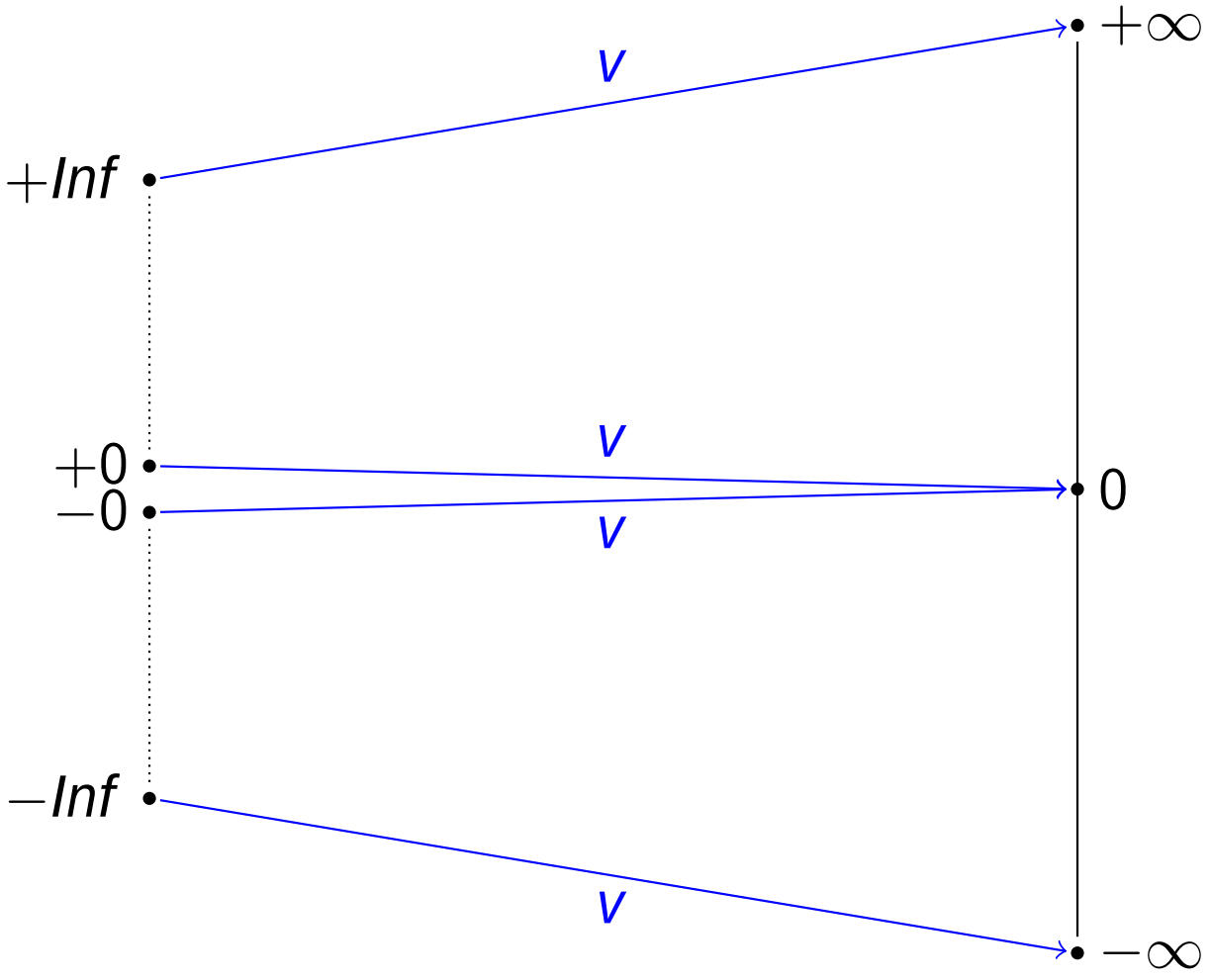
All conforming implementations of this standard shall provide the operations listed in this clause for all supported arithmetic formats, except as stated below. Each of the computational operations that return a numeric result specified by this standard shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that intermediate result, if necessary, to fit in the destination's format (see 4 and 7). Clause 6 augments the following specifications to cover ±0, ±∞, and NaN. Clause 7 describes default exception handling.

Upper and Lower Adjoints

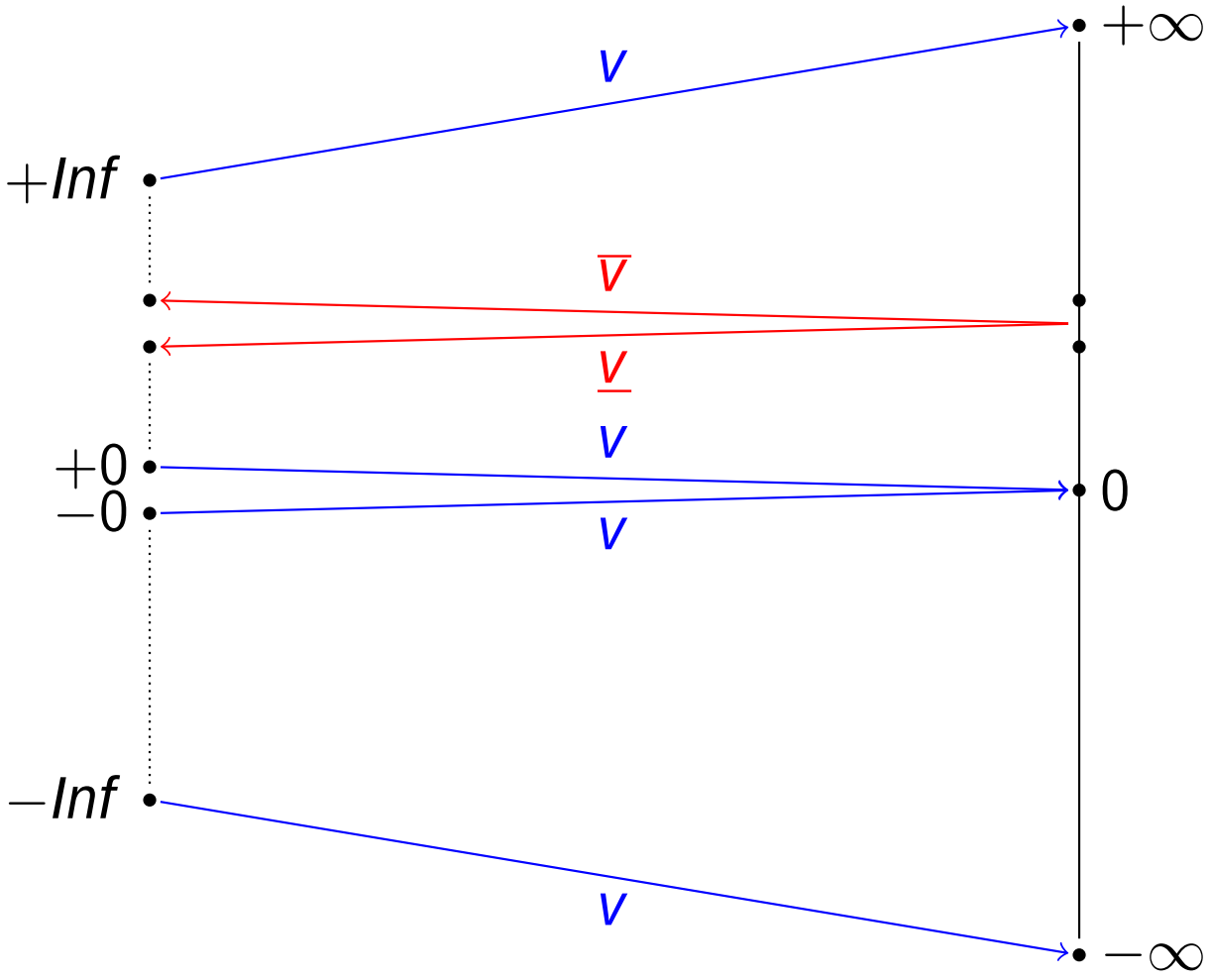




Upper and Lower Adjoints



Upper and Lower Adjoints



Rounding is (Just) Selecting Between Adjoints!

$$\mathrm{rnd}(v, \mathit{mode}, \mathit{sz}, r) = \overline{v}(r) \text{ or } \underline{v}(r)$$

This allows us to round to any format of float, bit-vectors, $\mathbb{Z}$, integer valued floats, ...

Operations

$$\mathrm{add}_{\varepsilon,\sigma}(rm, f, g) = \mathrm{rnd}(\mathrm{v}, rm, \mathrm{addSign}(rm, f, g), \mathrm{v}(f) + \mathrm{v}(g))$$

$$\mathrm{sub}_{\varepsilon,\sigma}(rm, f, g) = \mathrm{rnd}(\mathrm{v}, rm, \mathrm{subSign}(rm, f, g), \mathrm{v}(f) - \mathrm{v}(g))$$

$$\mathrm{mul}_{\varepsilon,\sigma}(rm, f, g) = \mathrm{rnd}(\mathrm{v}, rm, \mathrm{xorSign}(f, g), \mathrm{v}(f) * \mathrm{v}(g))$$

$$\mathrm{div}_{\varepsilon,\sigma}(rm, f, g) =$$

$$\begin{cases} \mathrm{neg}_{\varepsilon,\sigma}(\mathrm{rnd}(v, rm, \top, -(\mathrm{v}(f)/\mathrm{v}(g)))) & \mathrm{xorSign}(f, g) \\ \mathrm{rnd}(v, rm, \bot, \mathrm{v}(f)/\mathrm{v}(g)) & \neg\mathrm{xorSign}(f, g) \end{cases}$$

$$\mathrm{fma}_{\varepsilon,\sigma}(rm, f, g, h) =$$

$$\mathrm{rnd}(\mathrm{v}, rm, \mathrm{fmaSign}(rm, f, g, h), (\mathrm{v}(f) * \mathrm{v}(g)) + \mathrm{v}(h))$$

Limitations and Omissions

- No decimal floats
- Only one NaN (no signaling / quiet, no payload)
- No exceptions
- No attributes
- No trigonometric functions

Implementations

	Bit-blast	ACDL	Axiomatic
CVC4	(✓)	(✓)	
Z3	✓		
MathSAT	✓	✓	
Sonolar	✓		
Alt-Ergo			(✓)
CBMC	✓		

Help Needed!

- Correctness
- Examples
  (edge cases, tests, challenge problems)
- "Diamond free" circuits
  (multiply, divide, shift, float add, normalise)
- Elementary functions
- Floating-point remainder

Conclusions

1. Formalisation as input (axiomatic) vs. formalisation as specification (algebraic)
2. Rounding as choice of adjoints.
3. Have a specification (and implementations) of an SMT-LIB standard of floating-point.

Conclusions

1. Formalisation as input (axiomatic) vs. formalisation as specification (algebraic)
2. Rounding as choice of adjoints.
3. Have a specification (and implementations) of an SMT-LIB standard of floating-point.

Thank you for your time and attention.

Made using only Free Software