Bit-Exact Automated Reasoning About Floating-Point Arithmetic

Martin Brain

University of Oxford

July 4, 2018

What Do You Want To Know?

A Simple Program...

```
int takeNSamples (unsigned int n,
                  float start, float end) {

  if ((n == 0) || (end - start <= 0))
    return ERROR;

  float increment = (end - start) / (n + 1.0f);
  float location = start + increment;

  while (!(location >= end)) {
    sample(location);
    location += increment;
  }

  return DONE;
}
```

Want To Use Program Analysis Tools ...

... So Need An SMT Solver That Is ...

Bit-Exact	Must do *exactly* what the hardware does
Precise	Gives SAT / UNSAT
Model Generating	Gives a model if SAT
Automated	Ideally fast and "out of the box"

Other Use Cases

- ✓ Path feasibility / test-case generation
- ✓ Generation of special values
- ✓ Numerical instability
- ✓ (Language-level) Undefined behaviour
- ✓ Verification of hardware floating-point units
- ✓ Something about neural nets...
- ? Functional correctness of numerical code
- ? Automated numerical analysis

Let's Build a Number System!

Let's Build a Number System!

But what about over-flow?

Let's Build a Number System!

But what about over-flow? OK, but what is $\infty - \infty$?

Let's Build a Number System!

But what about over-flow? OK, but what is $\infty - \infty$?
That's cool but what about underflow?

Let's Build a Number System!

But what about over-flow? OK, but what is $\infty - \infty$?
That's cool but what about underflow?
Hold on; this can't be algebraically closed?

Let's Build a Number System!

But what about over-flow? OK, but what is $\infty - \infty$?
That's cool but what about underflow?
Hold on; this can't be algebraically closed?
Yeah but what are the actual values?

Floating-Point

Representing Rationals As...

$$-1^{sign} * base^{exponent} * significand$$

$$\begin{aligned} 7.28125 &= (2^2 + 2^1 + 2^0) + (2^{-2} + 2^{-5}) \\ &= 111.01001_2 \\ &= -1^0 * 2^2 * 1.1101001_2 \end{aligned}$$

Constant Relative Error?

For normal f is $\frac{f - prev(f)}{f} \leqslant c$?

Why is Reasoning About Floating-Point Hard?

Combinatorics

$$a + b$$

5 rounding modes
a is normal $\vee$ subnormal $\vee$ zero $\vee$ infinite $\vee$ NaN
b is normal $\vee$ subnormal $\vee$ zero $\vee$ infinite $\vee$ NaN

$$\Rightarrow 125 \text{ cases}$$

Rounding

$$\begin{aligned} (a+b)+c &\neq a+(b+c) \\ (a*b)*c &\neq a*(b*c) \\ a*(b+c) &\neq a*b+a*c \end{aligned}$$

Why is Reasoning About Floating-Point Software Hard?

Ranges Used vs. Ranges Specified

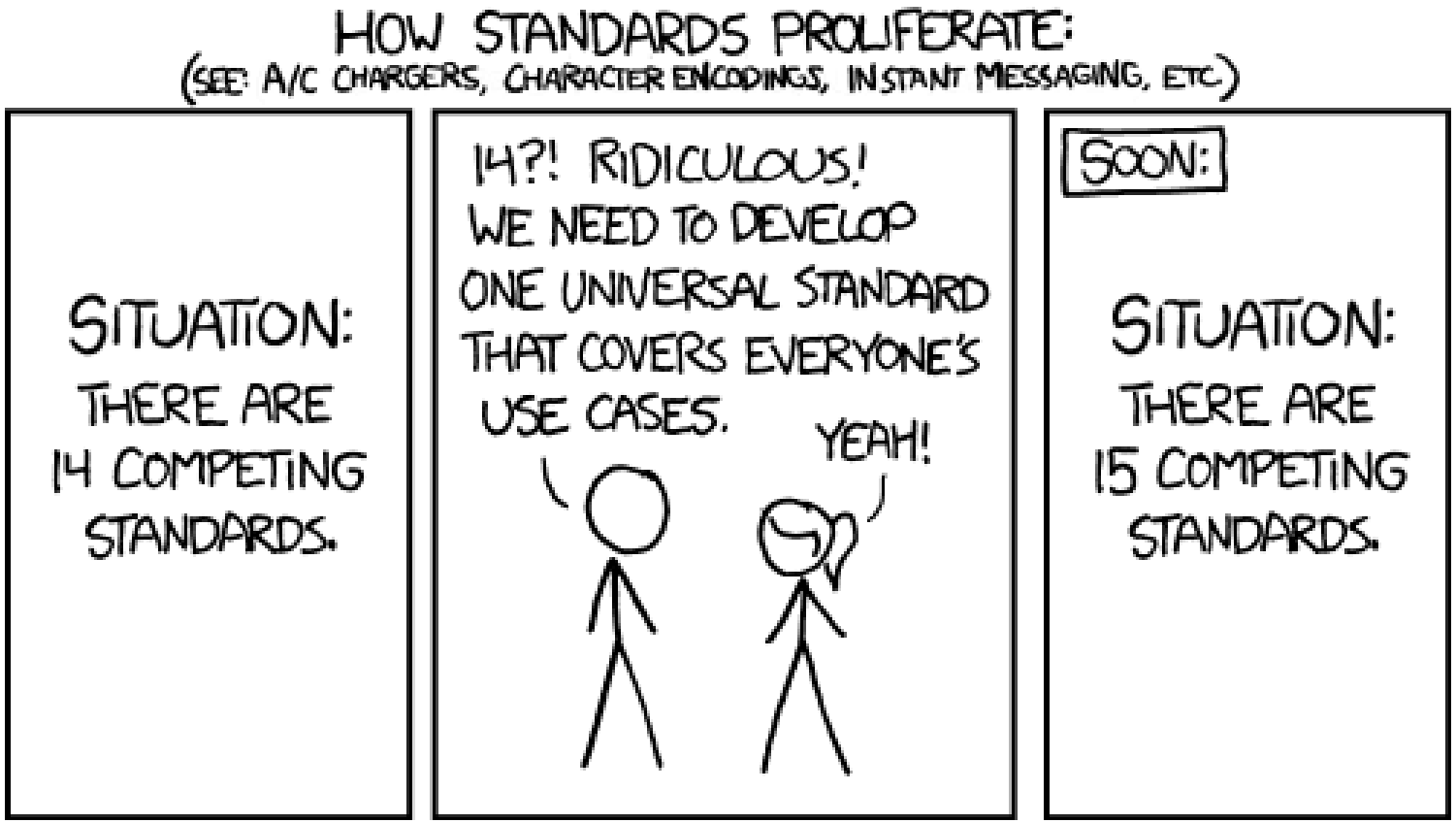
	Range	Data Bits
Likely speed in m/s	$[1 * 10^{-15}, 3 * 10^{8}]$	16
Positive range of `double`	$[1 * 10^{-324}, 1 * 10^{-308}]$	63

$[-1, 1]$ is half of all floats!

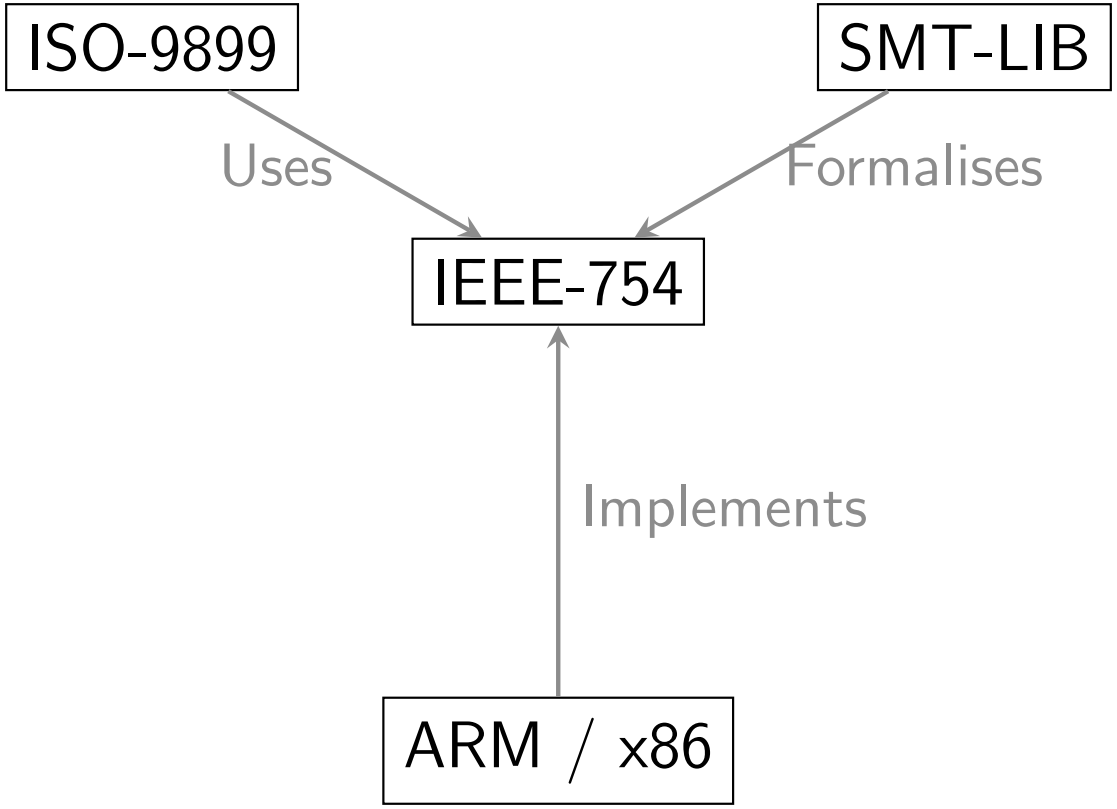
Specifications are "obvious"

"Obviously it doesn't work for ... but it doesn't need to."
"It won't ever be called with ... anyway."

Standards


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Standards


ISO-9899
SMT-LIB
Uses
Formalises
IEEE-754
Implements
ARM / x86

The SMT-LIB Initiative

http://smt-lib.org

- International initiative
- Rigorously standardise descriptions of theories for SMT
  Arithmetic ($\mathbb{Z}$ and $\mathbb{R}$), arrays, bit-vectors, floating-point, data-types
  (in preparation strings, sets, separation logic ...)
- Promote common syntax for SMT interactions
- Benchmarks
- Annual competition

How To Specify an SMT-LIB Theory

Fix a signature Σ ...

- In the theory files on the website `http://smt-lib.org/theories-FloatingPoint.shtml`
- Files have a formal(ish) syntax and semantics
- Intended as a specification for users / solvers

... and its interpretation (D, I)

- Given as (human readable) maths
- Most theories it is in the `:values` (D) and `:definition` (I) sections of the theory file.
- For floating-point it is a separate document `http://smt-lib.org/papers/BTRW15.pdf`

D : Domain of Interpretation

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***) $\cup \{-\infty, +\infty\} \cup$ **qNaN** $\cup$ **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 mathematical structure underpinning the arithmetic in this standard is the extended reals, that is, the set of real numbers together with positive and negative infinity. For a given format, the process of *rounding* (see 4) maps an extended real number to a *floating-point number* included in that format. A *floating-point datum*, which can be a signed zero, finite non-zero number, signed infinity, or a NaN (not-a-number), can be mapped to one or more *representations of floating-point data* in a format.

D : Domain of Interpretation

Level 1 $\mathbb{R}^{\dagger} = \mathbb{R} \cup \{+\infty, -\infty\}$

Level 2 $\mathbb{F}_{\varepsilon,\sigma} = \mathrm{F}_{\varepsilon,\sigma} \cup \{\mathrm{NaN}\}$

Level 3 -

Level 4 `(fp #b0 #b11..0 #b10...1)`

D : Examples of Constants

Examples

```
; 1.0f
(fp #b0 #b01111111 #b00000000000000000000000)

; +0
(fp #b0 #b00000000 #b00000000000000000000000)
(_ +zero 8 24)

; NaN
(fp #b1 #b11111111 #b11111111111111111111111)
(fp #b0 #b11111111 #b00000000000000000000001)
...
(_ NaN 8 24)
```

I : Interpretation of Symbols

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

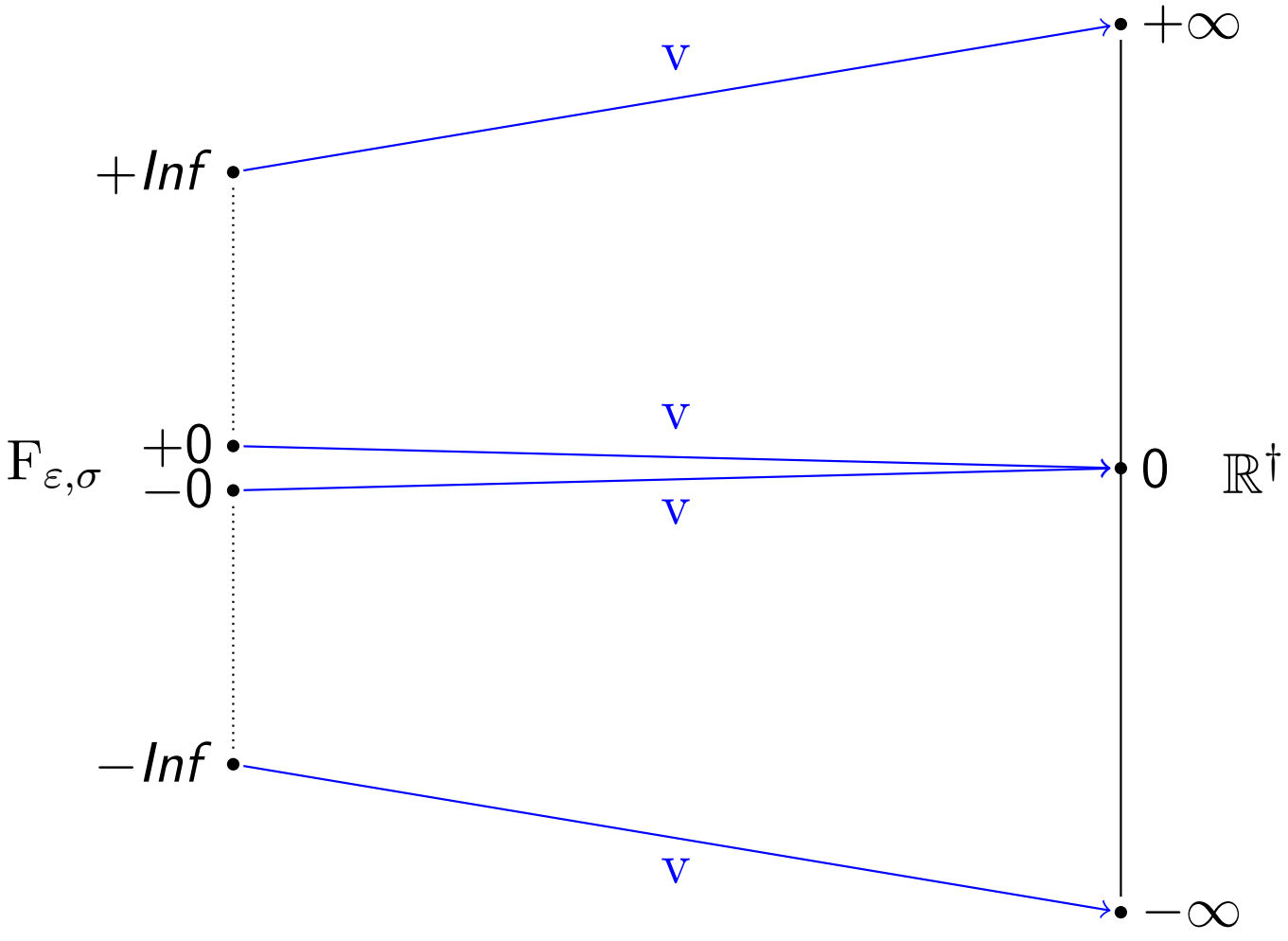
5. Operations

5.1 Overview

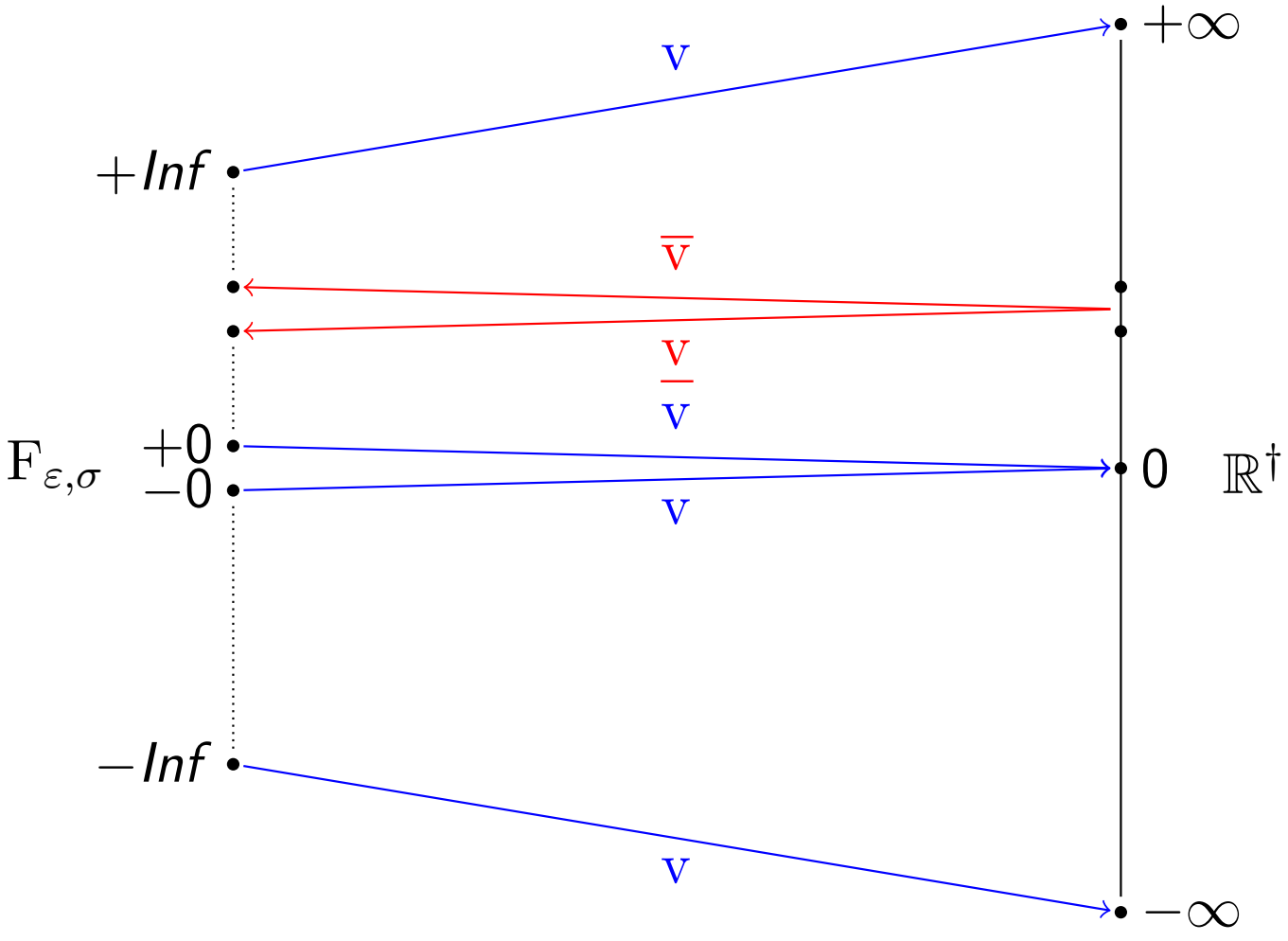
All conforming implementations of this standard shall provide the operations listed in this clause for all supported arithmetic formats, except as stated below. Each of the computational operations that return a numeric result specified by this standard shall be performed as if it first produced an intermediate 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.

I : Extended Reals

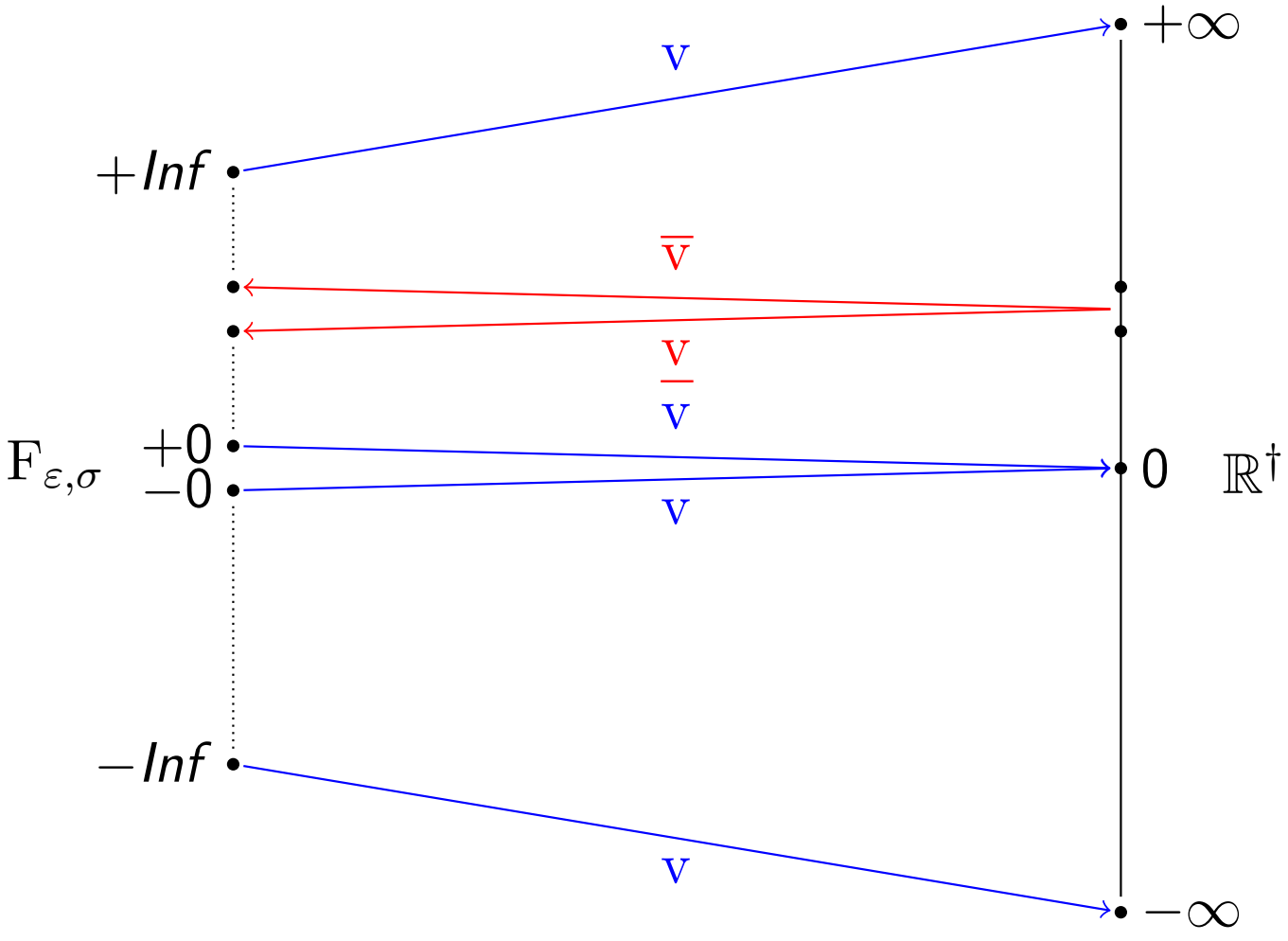
I : Extended Reals



I : Extended Reals



I : Extended Reals



$$\mathrm{rnd}(\mathrm{v}, rm, zsign, r) = \overline{\mathrm{v}}(r) \ \vee \ \mathrm{rnd}(\mathrm{v}, rm, zsign, r) = \underline{\mathrm{v}}(r)$$

I : Operations

Σ : Syntax

```
(fp.sub RoundingMode
        (_ FloatingPoint eb sb)
        (_ FloatingPoint eb sb)
        (_ FloatingPoint eb sb))
```

I : Semantics

$$
\begin{aligned}
[\![\texttt{fp.sub}]\!](rm, f, g) &= \mathrm{rnd}(\mathrm{v}, rm, \mathrm{subSign}(rm, f, g), \mathrm{v}(f) - \mathrm{v}(g)) \\
\mathrm{subSign}(rm, f, g) &= \begin{cases} \mathrm{isNeg}(f) \wedge \neg\mathrm{isNeg}(g) & rm \neq \mathrm{rtn} \\ \mathrm{isNeg}(f) \vee \neg\mathrm{isNeg}(g) & rm = \mathrm{rtn} \end{cases}
\end{aligned}
$$

Example

```
(fp.sub rnd end start)
```

I : Relations

Σ : Syntax

```
(fp.eq (_ FloatingPoint eb sb) (_ FloatingPoint eb sb) Bool)
(fp.leq (_ FloatingPoint eb sb) (_ FloatingPoint eb sb) Bool)
(fp.lt  (_ FloatingPoint eb sb) (_ FloatingPoint eb sb) Bool)
```

I : Semantics

$$
\begin{aligned}
[\![\texttt{fp.eq}]\!] &= \{(f,g) \in \mathrm{F}_{\varepsilon,\sigma} \times \mathrm{F}_{\varepsilon,\sigma} \mid \mathrm{v}(f) = \mathrm{v}(g)\} \\
[\![\texttt{fp.leq}]\!] &= \{(f,g) \in \mathrm{F}_{\varepsilon,\sigma} \times \mathrm{F}_{\varepsilon,\sigma} \mid \mathrm{v}(f) \leqslant \mathrm{v}(g)\} \\
[\![\texttt{fp.lt}]\!] &= \{(f,g) \in \mathrm{F}_{\varepsilon,\sigma} \times \mathrm{F}_{\varepsilon,\sigma} \mid \mathrm{v}(f) < \mathrm{v}(g)\}
\end{aligned}
$$

Example

```
(fp.eq start end)
(fp.leq (fp.sub rnd end start) (+zero 8 24))
```

Putting It All Together

From our program

```
float increment = (end - start) / (n + 1.0f);
```

In SMT-LIB

```
(declare-fun n     () (_ BitVec 32))
(declare-fun start () Float32)
(declare-fun end   () (_ FloatingPoint 8 24))
(define-fun  rnd   () RoundingMode RNE)

(define-fun increment () (_ FloatingPoint 8 24)
  (fp.div rnd
    (fp.sub rnd end start)
    (fp.add rnd
      ((_ to_fp_unsigned 8 24) rnd n)
      (fp #b0 #b01111111 #b00000000000000000000000))))
```

Let's Have A Break!

How Do We Come Up With Decision Procedure?

What Do We Know About Floating-Point?

How Do We Come Up With Decision Procedure?

What Do We Know About Floating-Point?

- Hardware implementations $\longrightarrow$ "Bit-blasting"
- Is weakly monotonic $\longrightarrow$ Interval
- Can be formalised $\longrightarrow$ Axiomatic
- Fast implementation $\longrightarrow$ Numeric

"Bit-Blasting"

Reduce to the Bit-Vector Theory

- Use circuits for a floating-point unit
- Simple to implement on top of a bit-vector solver
- Detailed development, computationally heavy
- Under and over approximation *of the problem*

Implementations

CBMC, Z3, MathSAT, SONOLAR, CVC4

Intervals

Weak Monotonicity

$$s < 0 \wedge a < b \Rightarrow a + s \leqslant b + s$$

so...

$$\mathtt{x} \in [x_l, x_h],\ \mathtt{y} \in [y_l, y_h],$$

$$\mathtt{(fp.add\ rnd\ x\ y)} \in [\mathtt{(fp.add\ RTN}\ x_l\mathtt{,}\ y_l\mathtt{)}, \mathtt{(fp.add\ RTP}\ x_h\mathtt{,}\ y_h\mathtt{)}]$$

Implementations

Propagate++	FPCS, COLIBRI
Search++	MathSAT-ACDCL, iSAT
AI	Fluctuat, Astrée, Polyspace, CodePeer
Error bound	Rosa, Daisy

Axiomatic

Full Eager Axiomatisation (in $\mathbb{R}$ and $\mathbb{Z}$)

- Isabelle, HOL, HOL Light, ACL2, PVS, Coq and Meta-Tarski
- Limited automation in ITP
- Decideability issues

Partial and Lazy Axiomatisation

Gappa	Heuristic instantiation of an axiomatisation
Gappa / Alt-Ergo	Gappa combined with Alt-Ergo
Why3	Eager partial axiomatisation of float over reals
Alt-Ergo	Lazy partial axiomatisation of float over reals
KLEE-FP	Rewriting based handling of equalities

Numerical

Fast Handling of Complete Assignments

- Hardware FPUs are cheap and fast!
- Can do various "black-box" search techniques
- Can't show UNSAT

Implementations

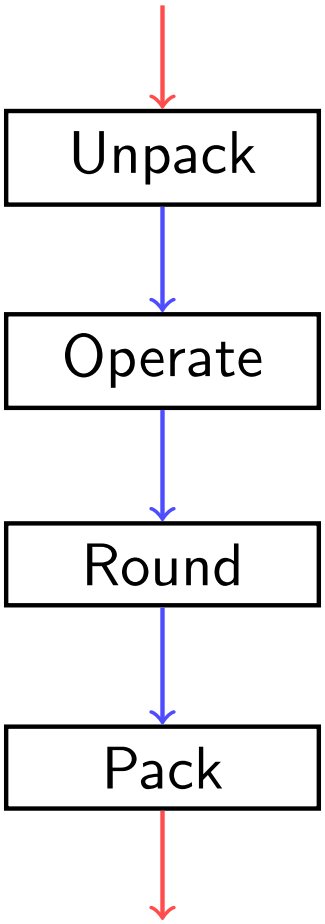
XSAT	Float-valued semi-definite solver
goSAT	Compile formula to executable code
JFS	External fuzzing tool

SymFPU

`https://github.com/martin-cs/symfpu/`

- C++ (header only) library of bit-vector encodings / circuits
- Parametric in bit-vector, propositional, rounding mode representations
- Reasonably well optimised (good circuit $\neq$ good encoding!)
- Correct for at least `Float16`, `Float32`, `Float64`, `Float128`

Basic FPU Data-path



Unpacked Formats (Level 2 representation)

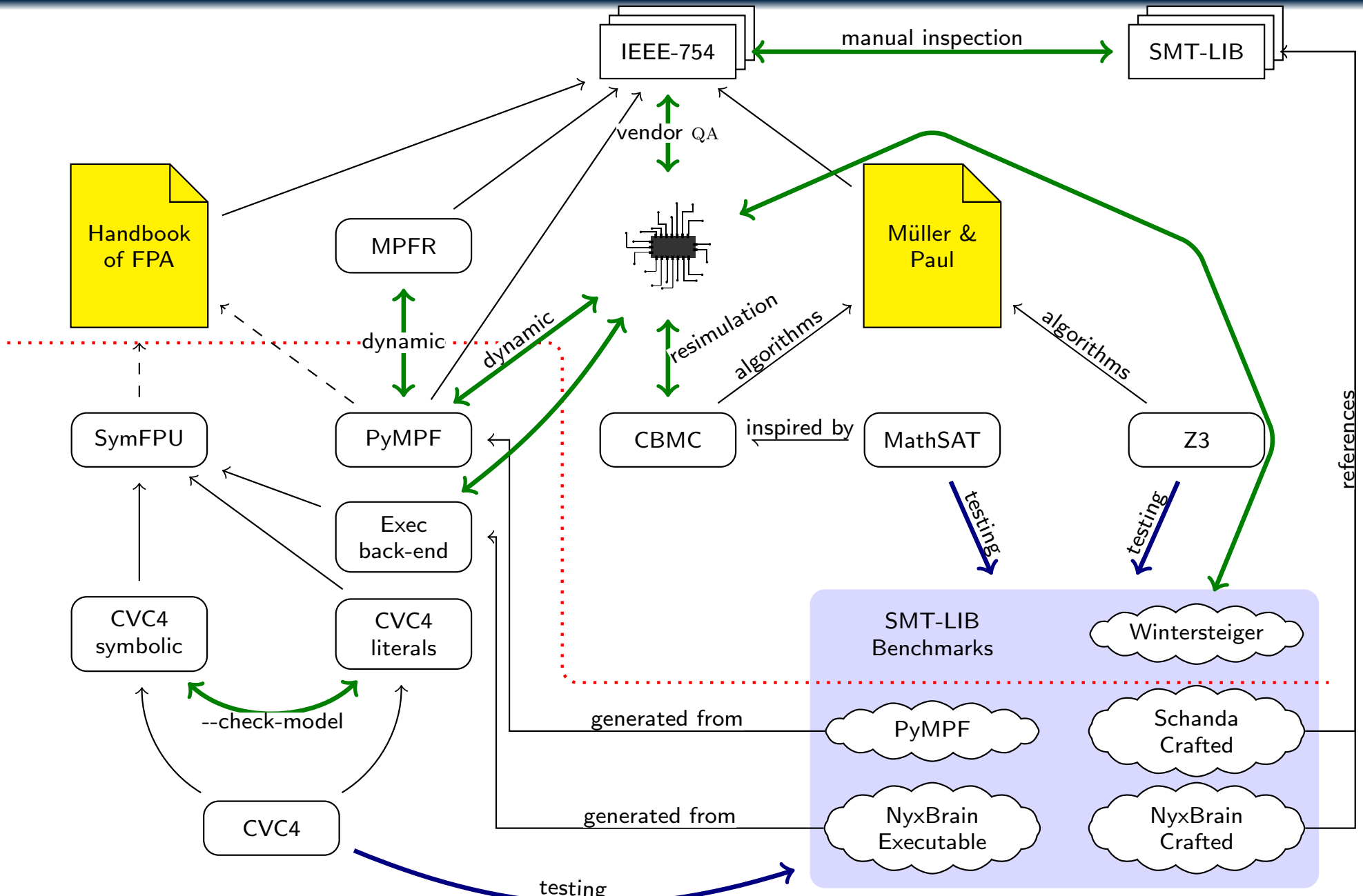
Options:

- Flags for NaN, ∞, 0, subnormals, ...
- Hidden-bit in significand
- Exponent biased or unbiased
- Handling of subnormals
- Lazy normalisation
- Redundant representations

Walkthrough

People say they want to see the encodings ...

Correctness



Performance

Benchmark	AE	Col	CVC4	goSAT	MS	MS-A	SON	Z3	Z3-SF	VBS
CBMC		63.0	55.6	9.3	50.0	**64.8**	38.9	44.4	46.3	81.5*
Schanda		82.8	**85.4**	1.5	66.7*	28.3*		82.3	81.8	95.5*
Griggio	0.9*	60.7	59.8	41.1	49.1	**68.2**	60.7	33.6	34.1	88.3*
Heizmann		14.0	**76.3**		58.5	27.5	2.9	40.6	34.3	89.9
Industrial 1	68.6*		**90.8**					65.4	62.5	91.3*
Industrial 1 (QF)		72.0	**99.2**		96.6	85.8		83.4	82.6	99.7
NyxBrain		99.8	**99.9**	34.2	95.4	95.0	99.2	99.9	99.8	>99.9
PyMPF		92.2	**99.7**	0.3	39.3	35.8		98.8	98.4	99.8
SPARK FP	64.1*		**84.6**					82.1	74.8	88.9*
SPARK FP (QF)		88.6	**94.9**		82.7	78.6		88.8	87.9	99.6
Wintersteiger	49.9*	✓	✓	13.9	85.8	85.8		✓	✓	✓

Percentage of problems solved in each category of benchmarks

Open Problems

- New approaches
- Mixed real / float decision procedures
- Mixed real / float / bit-vector decision procedures
- Trigonometric functions
- Scalability!

Conclusions

1. Tool support is vital for floating-point in critical software.
2. The theory of floating-point in the SMT-LIB standard.
3. There are "off-the-shelf" solvers.
4. Some interesting challenges.

Conclusions

1. Tool support is vital for floating-point in critical software.
2. The theory of floating-point in the SMT-LIB standard.
3. There are "off-the-shelf" solvers.
4. Some interesting challenges.

Thank you for your time and attention.

Made using only Free Software