Bit-Exact Automated Reasoning About Floating-Point Arithmetic

Martin Brain

University of Oxford

February 2, 2015

About Me...

- PhD at University of Bath in Logic Programming
- Co-author of Riposte, SPARK counter-example generator
- Post-doc researcher at University of Oxford
- Member of Daniel Kroening's CProver group
- CVC4 developer

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;

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

  return DONE;
}
```

Want To Use Program Analysis Tools ...

... So Need Theorem Proving 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"

Floating-Point

Representing Rationals As...

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

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

IEEE-754 (2008)

In IEEE-754

- Binary and decimal
- Formats
- Rounding
- Infinities, Not-a-Number
- Two zeros!
- Gradual underflow (subnormals)
- Arithmetic
- Comparison
- Exceptions (and handling)

Not In IEEE-754

- Sign and significand of NaN
- $max(+0, -0)$
  $min(+0, -0)$
- Exceptional conversion to `int`
- Math functions (sin, cos, exp)
- Reductions (sum, product, etc.)
- Expression evaluation

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

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

Applications

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

Satisfiability Modulo Theories (SMT)

Theory

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

Satisfiability of ϕ

ϕ a formula over Σ with $\Sigma' \subseteq \Sigma$
Is there M extension of M' such that:

$$M \models \phi$$

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
  (in preparation strings, data-types, sets ...)
- Promote common syntax for SMT interactions
- Benchmarks
- Annual competition

How To Specify a Theory

Fix a signature Σ' ...

- In the theory files on the website
- Files have a formal(ish) syntax and semantics
- Intended as a specification for users / solvers

... and its interpretation M'

- Given as (human readable) maths
- Most theories it is in the `:values` and `:definition` sections of the theory file.
- For floating-point it is a separate document

Sorts

Syntax

```
RoundingMode
(_ FloatingPoint eb sb)
Float16
Float32
Float64
Float128
```

Example

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

Domain of Interpretation

Syntax

```
(fp (_ BitVec 1) (_ BitVec eb) (_ BitVec i)
    (_ FloatingPoint eb sb))
```

Semantics

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

where

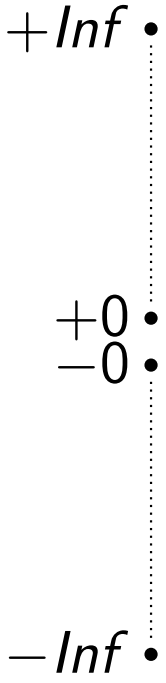
$$\begin{aligned}
\mathrm{F}_{\varepsilon,\sigma} &= \mathrm{FZ}_{\varepsilon,\sigma} \cup \mathrm{FS}_{\varepsilon,\sigma} \cup \mathrm{FN}_{\varepsilon,\sigma} \cup \mathrm{FI}_{\varepsilon,\sigma} \\
\mathrm{FZ}_{\varepsilon,\sigma} &= \{(s,e,m) \in B_{\varepsilon,\sigma} \mid e = \mathbf{0}_{\varepsilon},\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

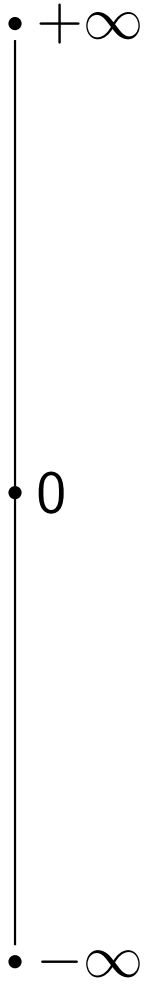
Examples of Constants

Examples

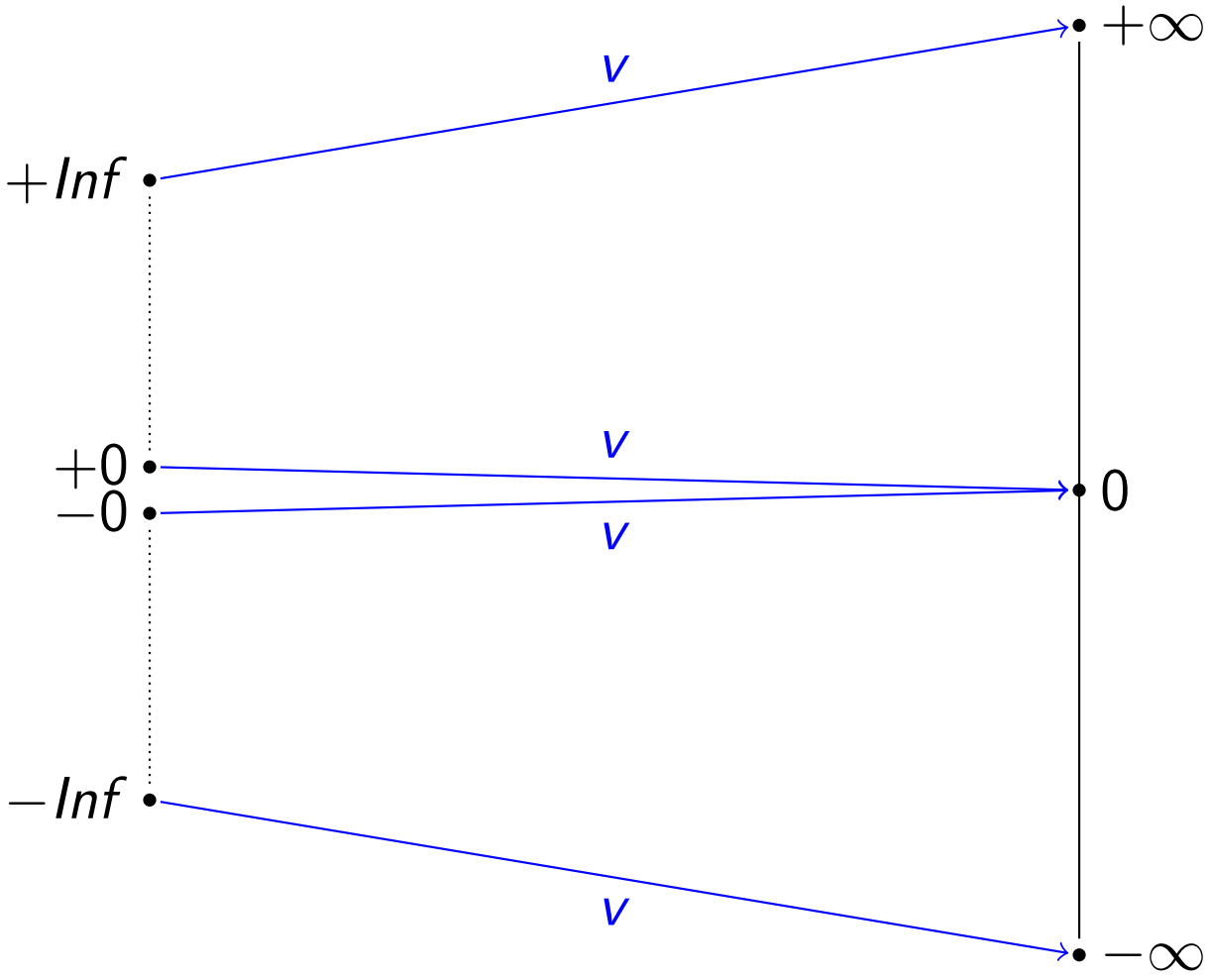
```
; 1.0f
(fp #b0 #b01111111 #b00000000000000000000000)

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

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

Extended Reals

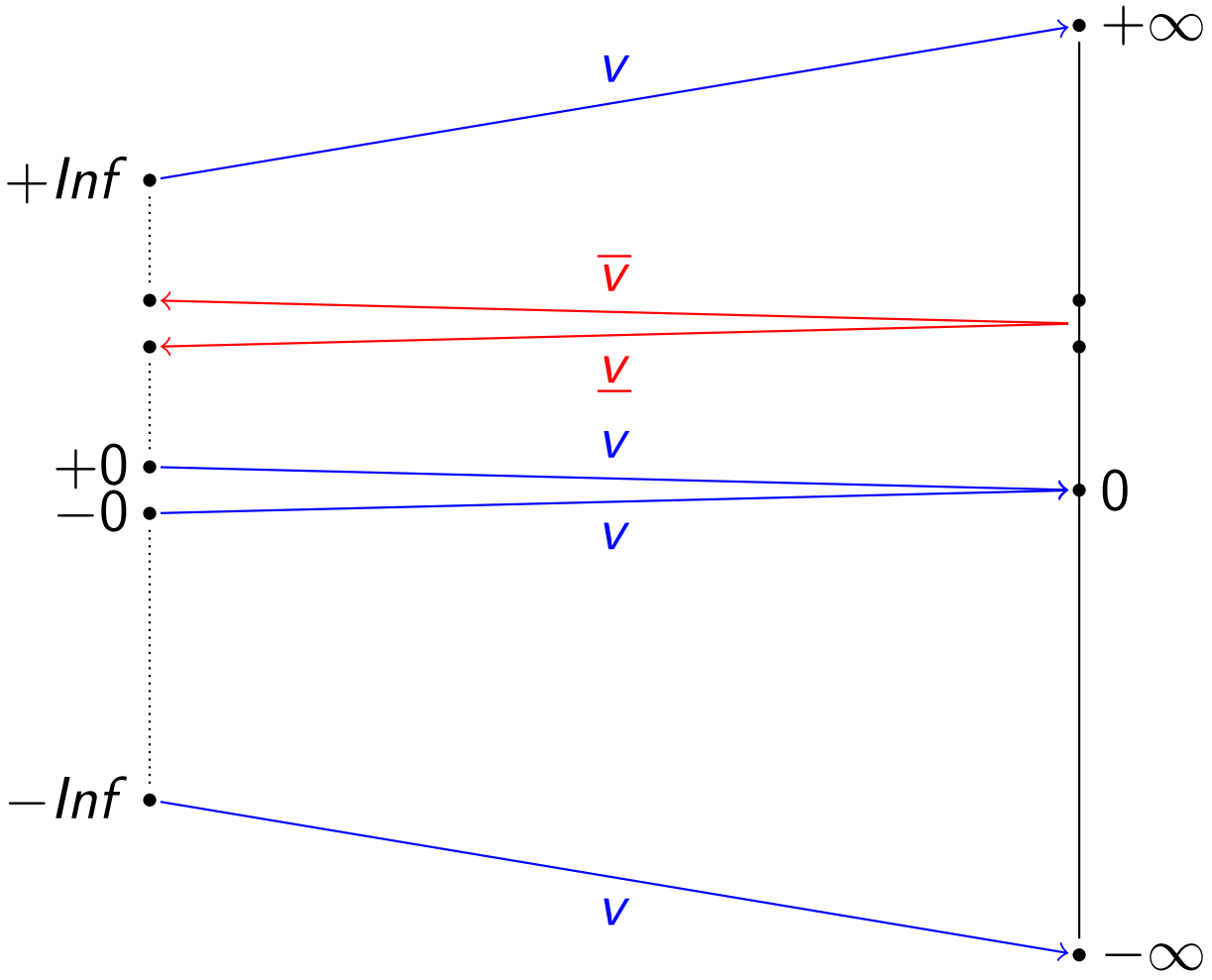




Extended Reals



Extended Reals



Operations

Syntax

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

Semantics

$$[\![\texttt{fp.sub}]\!](rm, f, g) = \mathrm{rnd}(\mathrm{v}, rm, \mathrm{subSign}(rm, f, g), \mathrm{v}(f) - \mathrm{v}(g))$$

$$\mathrm{subSign}(rm, f, g) = \begin{cases} \mathrm{isNeg}(f) \wedge \neg\mathrm{isNeg}(g) & rm \neq \mathrm{rtn} \\ \mathrm{isNeg}(f) \vee \neg\mathrm{isNeg}(g) & rm = \mathrm{rtn} \end{cases}$$

Example

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

Relations

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)
```

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))))
```

“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

Implementations

CBMC, Z3, MathSAT, SONOLAR, (CVC4 if I’m up late...)

Abstract Conflict Driven Learning

CDCL over non-Boolean Domains

- Use an *abstraction* to represent partial information about an interpretation (often intervals or constants)
- *Infer* as much as possible, then *branch*
- If a contradiction is found, *generalise* then *learn*

Implementation

ACDL	MathSAT, (CVC4 in development)
MCSAT	Z3-based nlsat
Constraint based	Absolute, FPCS, Popop (soon!)
Reals	dReal, iSAT

Others

Axiomatic

Why3	Axiomatisation of float over reals
Gappa	Heuristic instantiation of an axiomatisation
Gappa / Alt-Ergo	Gappa combined with Alt-Ergo

Others

ITP	Isabelle, HOL, HOL Light, ACL2, PVS, Coq
AI	Fluctuat, Astrée, Polyspace, CodePeer

Future

- Symfpu – verified and validated encodings to bit-vector
- Improved encodings
- Structural abstraction
- SMT-LIB benchmarks
- SMT-COMP floating-point track
- SV-COMP floating-point track

Open Problems

- Floating-point specific abstractions
- 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 is in the SMT-LIB standard.
3. There are "off-the-shelf" solvers.
4. Some interesting solver engineering challenges.

Conclusions

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

Thank you for your time and attention.

Made using only Free Software