Real Behavior of Floating Point Numbers

SMT 2017 | Bruno Marre, Bobot François, Zakaria Chihani

23 July 2017
COLIBRI (Bruno Marre)

- Started in 2000 for test case generation
- Used only as a library in PathCrawler and Gatel
- CP solver uses Eclipse Prolog
- Proprietary with the help of IRSN

- No test case that use NaN or infinities
- Only fp.eq, no =, only RNE, +0 = −0, only 32/64 bit
- integer modulo, real
COLIBRI (Bruno Marre)

- Started in 2000 for test case generation
- Used only as a library in PathCrawler and Gatel
- CP solver uses Eclipse Prolog
- Proprietary freeware for academic with the help of IRSN
- No test case that use NaN or infinities
- Only fp.eq, no =, only RNE, +0 = -0, only 32/64 bit
- integer modulo, real
Architecture

Propagation

Labelling Splitting

unsat

sat
Architecture

Propagation

Labelling Splitting

unsat

sat
Clear Semantic: \( x \oplus y = o(x + y) \)
Floating Points

- **Clear Semantic**: $x \oplus y = o(x + y)$
- **Few algebraic properties**: not associative, $x \oplus y = x \not\Rightarrow y = 0$
Floating Points

✅ Clear Semantic: \( x \oplus y = o(x + y) \)

❌ Few algebraic properties: not associative, \( x \oplus y = x \not\Rightarrow y = 0 \)

❌ Counter-intuitive: \( 0.1 \oplus \cdots \oplus 0.1 \neq 0.1 \otimes 10. = 1. \)
✓ Clear Semantic: $x \oplus y = o(x + y)$

✗ Few algebraic properties: not associative, $x \oplus y = x \not\sim y = 0$

✗ Counter-intuitive: $0.1 \oplus \cdots \oplus 0.1 \neq 0.1 \otimes 10. = 1.$

✗ State of the art: current bit-blasting doesn’t scale
Floating Points

✅ Clear Semantic: \( x \oplus y = o(x + y) \)

❌ Few algebraic properties: not associative, \( x \oplus y = x \not\Rightarrow y = 0 \)

❌ Counter-intuitive: \( 0.1 \oplus \cdots \oplus 0.1 \neq 0.1 \otimes 10. = 1. \)

❌ State of the art: current bit-blasting doesn’t scale

❌ Pervasives in programs
$X_i \in [1; 10] \implies X_0 \oplus X_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus X_5 \oplus X_6 \oplus X_7 \in [8; 80]$

Z3 : 3s
COLIBRI: < 0.1s (+0.25s)
\[ X_i \in [1; 10] \implies X_0 \oplus X_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus X_5 \oplus X_6 \oplus X_7 \in [8; 80] \]

Z3: 3s
COLIBRI: \(< 0.1s (+0.25s)\)

\[ X_i \in [1; 10] \implies X_0 \otimes X_1 \otimes X_2 \otimes X_3 \otimes X_4 \otimes X_5 \otimes X_6 \otimes X_7 \in [1; 10^8] \]

Z3: 31min
COLIBRI: \(< 0.1s (+0.25s)\)
Precise domain propagation:

\[ x \oplus y = 0.05 \implies x, y \in [-0.1259..; 0.175..] \]
Precise domain propagation:

\[ x \oplus y = 0.05 \implies x, y \in [-0.1259..; 0.175..] \]

0.05: 0x3fa99999999999999a
Precise domain propagation:
\[ x \oplus y = 0.05 \implies x, y \in [-0.1259; 0.175] \]

0.05: 0x3fa999999999999a

Distance graph on floating-point numbers
Distance graph on floating-point numbers

<table>
<thead>
<tr>
<th>x</th>
<th>IEEE-format, num(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\text{num}(x) - \text{num}(\text{fp.mul } 2 \times x) = 2^{52}
\]
Distance graph on floating-point numbers

<table>
<thead>
<tr>
<th>x</th>
<th>IEEE-format, num(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.</td>
<td>0</td>
</tr>
<tr>
<td>+1p - 1074</td>
<td>1</td>
</tr>
<tr>
<td>+1p - 1073</td>
<td>2</td>
</tr>
<tr>
<td>1.0</td>
<td>0x3ff000000000000000</td>
</tr>
<tr>
<td>2.0</td>
<td>0x4000000000000000</td>
</tr>
</tbody>
</table>

\[
\text{num}(x) - \text{num}(\text{fp.mul}_2 x) = 2^{52}
\]
## Distance graph on floating-point numbers

<table>
<thead>
<tr>
<th>$x$</th>
<th>IEEE-format, $\text{num}(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-2.0$</td>
<td>$-0x400000000000000000000000$</td>
</tr>
<tr>
<td>$-1.0$</td>
<td>$-0x3ff00000000000000000000$</td>
</tr>
<tr>
<td>$-1p - 1073$</td>
<td>$-2$</td>
</tr>
<tr>
<td>$-1p - 1074$</td>
<td>$-1$</td>
</tr>
<tr>
<td>$0.0$</td>
<td>$0$</td>
</tr>
<tr>
<td>$+1p - 1074$</td>
<td>$1$</td>
</tr>
<tr>
<td>$+1p - 1073$</td>
<td>$2$</td>
</tr>
<tr>
<td>$1.0$</td>
<td>$0x3ff00000000000000000000$</td>
</tr>
<tr>
<td>$2.0$</td>
<td>$0x400000000000000000000000$</td>
</tr>
</tbody>
</table>

$$\text{num}(x) - \text{num}(\text{fp.mul }_2 x) = 2^{52}$$
### Distance graph on floating-point numbers

<table>
<thead>
<tr>
<th>$x$</th>
<th>IEEE-format, $\text{num}(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-2.0$</td>
<td>$-0x40000000000000000000000000000$</td>
</tr>
<tr>
<td>$-1.0$</td>
<td>$-0x3ff00000000000000000000000000$</td>
</tr>
<tr>
<td>$-1p - 1073$</td>
<td>$-2$</td>
</tr>
<tr>
<td>$-1p - 1074$</td>
<td>$-1$</td>
</tr>
<tr>
<td>$-0.$</td>
<td>$0$</td>
</tr>
<tr>
<td>$0.$</td>
<td>$0$</td>
</tr>
<tr>
<td>$+1p - 1074$</td>
<td>$1$</td>
</tr>
<tr>
<td>$+1p - 1073$</td>
<td>$2$</td>
</tr>
<tr>
<td>$1.0$</td>
<td>$0x3ff00000000000000000000000000$</td>
</tr>
<tr>
<td>$2.0$</td>
<td>$0x400000000000000000000000000000$</td>
</tr>
</tbody>
</table>

$\text{num}(x) - \text{num}(\text{fp.mul}_2 x) = 2^{52}$
Distance graph on floating-point numbers

\[ x \in [1; 10], \text{fp.mul RNE} \times 2 = y \]

\[ \{2^{52}\} \]

\[ w \in [1; 10], \text{fp.add RNE} w 3 = z \]

\[ [\text{num}(13) - \text{num}(10); \text{num}(4) - \text{num}(1)] \]
Precise domain propagation:
\[ x \oplus y = 0.05 \implies x, y \in [-0.1259..; 0.175..] \]
0.05: 0x3fa99999999999a

Distance graph on floating-point numbers

Monotonic functions:
\[ o(f(x)) < o(y) \implies o(x) \leq o(f^{-1}(o(y))) \]
Precise domain propagation:
\[ x \oplus y = 0.05 \implies x, y \in [-0.1259..; 0.175..] \]
0.05: \texttt{0x3fa999999999999a}

Distance graph on floating-point numbers

Monotonic functions:
\[ o(f(x)) < o(y) \implies o(x) \leq o(f^{-1}(o(y))) \]

Instantiated for many functions
COLIBRI: Floating Point

- Precise domain propagation:
  \[ x \oplus y = 0.05 \implies x, y \in [-0.1259..; 0.175..] \]
  0.05: 0x3fa999999999999a

- Distance graph on floating-point numbers

- Monotonic functions:
  \[ o(f(x)) < o(y) \implies o(x) \leq o(f^{-1}(o(y))) \]

- Instantiated for many functions

- Linearization of constraints for simplex
Interesting and Simple Real Examples

```c
/*@ requires 0 \leq x \leq 1000;
    requires 0 \leq y \leq 1000;
    ensures 0 \leq \result \leq 1;  @*/
double x_normalisation(double x, double y){
    return x/sqrt(x*x + y*y);
}
```

COLIBRI: Example of Reasoning

\[0 \leq x, y \leq 1000 \implies \sqrt{x^2 + y^2} \geq x?\]
\[ 0 \leq x, y \leq 1000 \implies \sqrt{x^2 \oplus y^2} \geq x \ ? \]

\[ o \left( \sqrt{o(x^2) + o(y^2)} \right) < x \]

\[ o(x^2) + o(y^2) \leq o(x^2) \]

\[ o(x^2) + o(y^2) = o(x^2) \]

\[ o \left( \sqrt{o(x^2)} \right) < x \]

\[ x < x \text{ if } o(x^2) \text{ is normalized} \]

\[ o(x^2) \text{ is denormalized} \]

\[ x \text{ the minimum of the remaining values is a solution} \]
0 \leq x, y \leq 1000 \implies \sqrt{x^2 \oplus y^2} \geq x ?

o \left( \sqrt{o(x^2) + o(y^2)} \right) < x

o(x^2) + o(y^2) \leq o(x^2)
o(x^2) + o(y^2) = o(x^2)
o \left( \sqrt{o(x^2)} \right) < x

x < x \text{ if } o(x^2) \text{ is normalized}
o(x^2) \text{ is denormalized}
x \text{ the minimum of the remaining values is a solution}

There is a counter-example!
Interesting and Simple Real Examples: Corrected

```c
/*@ requires 0.0001 \leq x \leq 1000;
requires 0.0001 \leq y \leq 1000;
ensures 0 \leq \text{result} \leq 1; */

double x_normalisation(double x, double y){
    return x/sqrt(x*x + y*y);
}
```
procedure User_Rule_7 (X, Y, Z, A : Float;
    Res : out Boolean)
is
begin
    pragma Assume (Z ≥ 0.0);
    pragma Assume (X ≥ Y);
    pragma Assume (Y ≥ Z);
    pragma Assume (X > Z);
    pragma Assume (A ≥ 1.0);
    Res := (X − Y) / (X − Z) ≤ A;
    pragma Assert (Res);  -- valid
end User_Rule_7;
Other Examples: From SPARK User Rule

\[ A \leq \frac{X \oplus Y}{X \oplus Z} \leq B \quad \text{with} \quad \ldots \]

\[ \sqrt{X^2 \oplus Y^2} \leq X \quad \text{with} \quad \ldots \]

\[ \frac{X}{\sqrt{X^2 \oplus Y^2}} \leq 1 \quad \text{with} \quad \ldots \]
For \( t \) a normal positive number with double precision:

\[
o(t)
\]
For $t$ a normal positive number with double precision:

$$(1 - \frac{1}{2^{52} - 1}) \cdot t \leq o(t) \leq (1 + \frac{1}{2^{52} + 1}) \cdot t.$$
Linearization [Belaid2012]

For $t$ a normal positive number with double precision:

$$(1 - \frac{1}{2^{52} - 1}) \cdot t \leq o(t) \leq (1 + \frac{1}{2^{52} + 1}) \cdot t.$$ 

$$(0. \leq_f x \leq_f 10.0) \land (0. \leq_f y \leq_f 10.0) \Rightarrow ((x \oplus y) \ominus x) \ominus y \leq_f 0.0001$$
Linearization [Belaid2012]

For $t$ a normal positive number with double precision:

$$(1 - \frac{1}{2^{52} - 1}) \cdot t \leq o(t) \leq (1 + \frac{1}{2^{52} + 1}) \cdot t.$$ 

$$(0. \leq_f x \leq_f 10.0) \land (0. \leq_f y \leq_f 10.0) \Rightarrow \quad \circ(\circ(\circ(x + y) - x) - y) \leq_f 0.0001$$
- High-level view of bitvectors
- New propagations for integers ↔ bitvectors
Interreductions

Diagram:

- Int/BV
- Δ
- D
- FP
- Real
- Arrows indicate flow or relationship between different domains.
\[ x, y \in [1; 1000], \text{fp.to_sbv}_x = w, \text{fp.to_sbv}_y = z \]
Casts

\[ x, y \in [1; 1000], \text{fp.to_sbv}_x = w, \text{fp.to_sbv}_y = z \]
Grigio and Schanda

![Graph showing performance comparison of COLIBRI, no simplex, no delta, MathSAT, ACDCL, and Z3 over time.](image-url)
Future Work

- Look at the unsolved benchmarks
Future Work

- Look at the unsolved benchmarks
- More confidence in the propagation and rewrite rules
Future Work

- Look at the unsolved benchmarks
- More confidence in the propagation and rewrite rules
- Uninterpreted functions and quantifiers
Future Work

- Look at the unsolved benchmarks
- More confidence in the propagation and rewrite rules
- Uninterpreted functions and quantifiers
- MCsat
Future Work

- Look at the unsolved benchmarks
- More confidence in the propagation and rewrite rules
- Uninterpreted functions and quantifiers
- MCsat
- Reduce the loading time...
Theorem

Let $D, E \subseteq \mathcal{R}$, $f : D \mapsto E$ and $f^{-1} : E \mapsto D$ such that

- $\forall x : D$, $f^{-1}(f(x)) = x$
- $f$ increasing

We have

- $\forall x \in D$, $o(y) \in E$, $o(f(x)) < o(y) \implies o(x) \leq o(f^{-1}(o(y)))$
- $\forall x \in D$, $y \in E$, $o(f(x)) < o(f(y)) \implies x < y$

Instantiated for many functions in COLIBRI's DBM
/@ ensures \result \leq (\texttt{double}) 1; @*/
double test2() {
    double x = read_sensor();
    /@ assert (\texttt{double}) 0 \leq x \leq (\texttt{double}) 1000; @*/
    double y = read_sensor();
    double z = read_sensor();

    x = x * x + z * z + y * y + 1;

    if (z \leq y) {
        return (x - y) / (x - z);
    } else {
        return (x - z) / (x - y);
    }
}
The Problems