Reasoning about floating-point arithmetic with ACDCL
Unifying Abstract Interpretation and Decision Procedures

Daniel Kroening
(joint work with Leopold Haller, Vijay D’Silva, Michael Tautschnig, Martin Brain)

9 January 2013
Leopold Haller

Vijay D’Silva

Michael Tautschnig

+ Martin Brain (no photo)
References

• TACAS 2012: paths in floating-point programs with intervals
• POPL 2013: Framework
• VMCAI 2013: DPLL(T)
• FMCAD 2012: Learning for intervals
• SAS 2012: propositional SAT
Presentation Outline

Part I

Existing approaches to FP - Verification

- Manual, Semi-automated
- Decision Procedures
- Abstract Interpretation

Part II

- Decision Procedures
- Precise
- Scalable
- Abstract Interpretation

Abstract Satisfiability

Our research
Part I
IEEE754 Floating Point Numbers

**Example 1**

\[
\begin{align*}
0 & \quad 00000111 \quad 11000000000000000000000000000000 \\
+ & \quad 7 \quad 0.75
\end{align*}
\]

\[+1.75 \times 2^{(7-127)} = +1.316554 \times 10^{-36}\]

**Special values:** \(-0, +0, -\infty, \infty, NaN\)
The Pitfalls of FP

I

```c
if(x < y)
...
else if(x > y)
...
else assert(x == y);
```

II

```c
if(x > 0)
{
    for(float sum = 0; sum <= N; sum+=x)
        ...

    // does the loop terminate?
}
```

III

```c
float r1 = a+b;
float r2 = b+c;

r1+= c; r2 += a;
assert(r1 == r2);
```

IV

```c
float r1 = a+b;
float r2 = a+b;

assert(r1 == r2);
```

V

```c
bool b = false;

if(f < 1)
    b = true;

if(!b)
    assert(f >= 1);
```
Is this program correct?
(We will ignore the case x=NaN)
What does correctness mean?

Three possible meanings:

- Result is sufficiently close to the real number result
- Result is sufficiently close to the sine function
- The assertion cannot be violated
How can we check correctness?

Manual

Abstract Interpretation  Decision Procedures
Requires experts, expensive, powerful

Abstract Interpretation

Manual

Decision Procedures
Abstract Interpretation

- Instead of exploring all executions, explore a single abstract execution
- Abstract execution contains all concrete executions!
- Highly efficient and scalable, but imprecise

Error states do not overlap abstract representation, hence program is safe.
An abstract interpreter modularly uses operations provided by an abstract domain. Changing the domain changes the analysis.

### Example

<table>
<thead>
<tr>
<th>Signs domain</th>
<th>Constants domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>{+,-} \cup {?}</td>
<td>{c \mid c \in \text{FP}} \cup {?}</td>
</tr>
</tbody>
</table>

```c
float y = 5;
if(x > 0) {
  float z = x*y;
  assert(z > 0);

  y = +
  x = +
  z = +

  safe!
}
```

```
  y = 5
  x = ?
  z = ?
```

Possibly unsafe
An abstract interpreter modularly uses operations provided by an abstract domain. Changing the domain changes the analysis.

Example

```c
int x, y;

if (y < 0)
{
    x = y;
}
else
{
    y++;,
    x = 5;
}

assert(x < 6);
```

Interval Domain

\[
\{[l, u] \mid l, u \in Int\}
\]

\[
x, y \in [\text{min}(Int), \text{max}(Int)]
\]

\[
x \in [5, 5], y \in [\text{min}(Int), \text{max}(Int)]
\]

\[
x \in [\text{min}(Int), 5], y \in [\text{min}(Int), \text{max}(Int)]
\]
Floating Point Intervals

\{[l, u] \mid l, u \in FP\} \cup \{?\}

Abstract Interpretation

```c
#define HALFPI 1.57079632679f

float sine_approx(float x)
{
    if(x <= -HALFPI || x >= HALFPI);
        return 0.0f;
    result = x - (x*x*x)/6.0f;
    result += (x*x*x*x*x)/120.0f;
    result += (x*x*x*x*x*x*x)/5040.0f;
    assert(result <= 1.01 && result >= -1.01);
    return 0;
}
```

$x \in [-1.570796, 1.570796]$

\(result \in [-2.216760, 2.216760]\)

\(result \in [-2.296453, 2.296453]\)

\(result \in [-2.301135, 2.301135]\)

Potentially unsafe
Astrée Abstract Interpreter

- Mature abstract interpreter by Cousot et. al
- Large number of domains
- Sold and supported by Absint GmbH
- Successful in proving correct large avionics control software: 100k lines of code in 1h -> highly scalable
- Various domains for floating point analysis:

Original traces  Ellipses  Octagons  Intervals
Abstract Domains for Floating Point

- Abstract domains are typically formulated over the real or rational numbers
- Numeric domains rely on mathematical properties such as associativity which do not hold over floating point numbers

\[(a + b) + c = a + (b + c)\]

- Solution (Mine 2004): Interpret operations over floating point numbers as real number operations + error terms

```plaintext
double d;
float f1,f2;
f1 = (float) d;
f2 = f1*f2;
```

```plaintext
real d;
real f1, f2;
f1 = d + round_error(FLOAT_CAST,d);
f2 = f1*f2 + round_error(FLOAT_MULT, f1,f2);
```
The efficiency of abstract interpreters comes at the cost of precision. Imprecision is accumulated from three sources:

- **Statements**
  
  \[
  x \in [-5, 5] \quad y = x \ast x; \quad y \in [-25, 25] \\
  x \in [0, 1] \quad y = x; \quad x, y \in [0, 1]
  \]

- **Control-flow**
  
  \[
  \text{if}(y < 0) \quad x = 1; \\
  \text{else} \quad x = -1; \\
  x \in [-1, 1]
  \]

- **Loops**
  
  \[
  x, y \in [1, 1] \quad \textbf{while}(x < 100000) \quad x \in [100001, \max(Int)] \\
  \{ \text{x++; y++;} \} \quad y \in [\min(Int), \max(Int)]
  \]
Imprecision in Abstract Interpretation

- For efficiency reasons, most numeric abstract domains are convex

Original traces  Ellipses  Octagons  Intervals

Convex polyhedra  Zonotope

Example 1.
Imprecision in Abstract Interpretation

What if convex abstractions are too weak?

Very common scenario

```plaintext
if(*)
x = 1;
else
x = -1;
assert(x != 0);
```
Conclusion:

- Very scalable
- Imprecise
- Precise results require experts and research effort
- Expert created domains are moderately reusable
- Feasible for programs with homogenous structure and behaviour (success in avionics)
References

Floating point abstract domains

A. Chapoutot. Interval slopes as a numerical abstract domain for floating-point variables. SAS 2010

L. Chen, A. Miné and P. Cousot. A sound floating-point polyhedra abstract domain. APLAS 2008

A. Miné. Relational abstract domains for the detection of floating-point run-time errors. ESOP 2004

L. Chen, A. Miné, J. Wang and P. Cousot. An abstract domain to discover interval Linear Equalities. VMCAI 2010


K. Ghorbal, E. Goubault and S. Putot. The zonotope abstract domain Taylor I. CAV 2009

B. Jeannet, and A. Miné. Apron: A library of numerical abstract domains for static analysis. CAV 2009

D. Monniaux. Compositional analysis of floating-point linear numerical filters. CAV 2005

J. Feret. Static analysis of digital filters. ESOP 2004


E. Goubault and S. Putot. Weakly relational domains for floating-point computation analysis. NSAD 2005

E. Goubault. Static analyses of the precision of floating-point operations. SAS 2001
References

Industrial Case Studies

E. Goubault, S. Putot, P. Baufreton, J. Gassino. Static analysis of the accuracy in control systems: principles and experiments. FMICS 2007


J. Souyris and D. Delmas. Experimental assessment of Astrée on safety-critical avionics software. SAFECOMP 2007

J. Souyris. Industrial experience of abstract interpretation-based static analyzers. IFIP 2004

P. Cousot. Proving the absence of run-time errors in safety-critical avionics code. EMSOFT 2007

FP Static Analysers


P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and Xavier Rival. The ASTRÉÉ analyzer. ESOP 2005

E. Goubault, M. Martel and S. Putot. Asserting the precision of floating-point computations: a simple abstract interpreter. ESOP 2002
Requires experts, expensive, powerful

Abstract Interpretation
Scalable and efficient.
Precise analysis requires experts

Decision Procedures
Decision Procedures

- Precisely explore a large set of program traces
- For efficiency, represent problem symbolically as satisfiability of a logical formula

Program traces

Program is safe exactly if $isTrace(t) \land error(t)$ is satisfied by some t
Propositional formula:  \( \varphi = (a \lor \neg b) \land (\neg a \lor b) \land \neg b \)

Is there an assignment to \( a, b \) that makes the formula true?
Why are SAT solvers so efficient

Probe for solution \rightarrow Learn from failure \rightarrow failure

- SAT solvers learn from failure
- SAT solvers spot relevance
Example

```c
int foo(int a, int b, bool c) {
    int result;
    if(c)
        result = a/b;
    else
        result = a*b;
    if(a>0 && b>0)
        assert(result >= 0);
}
```

\[
c \rightarrow (r = a/32b) \\
\land \neg c \rightarrow (r = a \times 32 \ b) \\
\land a > 0 \land b > 0 \land r < 0
\]

Can be translated to propositional logic using divider and multiplier circuits

The formula evaluates to true under the following assignment:

\[
a, b \mapsto 123456789 \\
r \mapsto -1757895751 \\
c \mapsto \text{false}
\]

Counterexample!
Bounded Model Checking

Loops require unrolling before translation

```c
int foo(int *a)
{
    int sum;
    for(int i = 0; i < N; i++)
        sum+=a[i];
    assert(sum > 0);
    return sum;
}
```

```c
int foo(int *a)
{
    int sum;
    int i = 0;
    if(i < N)
    {
        sum += a[i];
        if(++i < N)
        {
            sum += a[i];
            if(++i < N)
            {
                ...
            }
        }
    }
    assert(sum > 0);
    return sum;
}
```

If the loop does not have a known fixed bound, the result is unrolled up to a chosen depth.
Bounded Model Checking

Program has bug, counter-example is returned

C/C++ Source → parse → parse tree → CFG → formula → flattening

Decision Procedure

Unsatisfiable

Satisfiable

Program has bug, counter-example is returned
FP support in CBMC (2008)

- CBMC implements **bit-precise reasoning** over floating-point numbers using a propositional encoding
- Uses IEEE-754 semantics with support for various rounding-modes
- Allows proofs of **complex, bit-level** properties

```c
int main()
{
  union {
    int i;
    float f;
  } u;

  u.f += u.i + 1;

  assert(u.i != 0);
}
```
Scalability of Propositional Encoding

- Floating-point arithmetic is flattened to propositional logic
- Requires instantiation of large floating point arithmetic circuits

```c
for(int i = 0; i < N; i++)
{
  f *= f;
}
```

<table>
<thead>
<tr>
<th>N</th>
<th>Nr. Variables</th>
<th>Memory use</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>~130000</td>
<td>~90MB</td>
</tr>
<tr>
<td>10</td>
<td>~260000</td>
<td>~180MB</td>
</tr>
</tbody>
</table>

- Resulting formulas are hard for SAT solvers and take up large amounts of memory
Related work

Constraint satisfaction

C. Michel, M. Rueher and Y. Lebbah: Solving constraints over floating-point numbers. CP2001

B. Botella, A. Gotlieb and C. Michel: Symbolic execution of floating-point computations. STVR2006

SMT

P. Ruemmer and T. Wahl. An SMT-LIB theory of binary floating-point arithmetic. SMT 2010

A. Brillout, D. Kroening and T. Wahl. Mixed abstractions for floating point arithmetic. FMCAD 2009

R. Brummayer and A. Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. TACAS 2009

Incomplete Solvers

Requires experts, scalable, precise

Abstract Interpretation
Scalable.
Precision requires experts

Decision Procedures
Precise.
Scalability requires experts
Conclusion Part I

Automatic

Scalable

Theorem proving

Precise

Abstract interpretation

Decision procedures

Abstract Interpreter

Safe

Decision Procedures

Bug

?
Questions so far?
Part II
We are interested in techniques that are
- scalable
- sufficiently precise to prove safety
- fully automatic

**Central insight:**
Modern decision procedures are abstract interpreters!
Manually adjusting analysis precision by abstract partitioning

```c
void foo(int x)
{
    int y;
    if(x < 0)
        y = 1;
    else
        y = -1;  // y ∈ [-1, 1]
    assert(y != 0);
}

Potentially unsafe!
```

```c
void foo_precise(int x)
{
    if(x < 0)
        foo(x);
    else
        foo(x);
}
```

Safe!
How do we find the partition automatically?
SAT solving by example

SAT solvers accept formulas in conjunctive normal form

\[ \varphi = (p \vee \neg q) \land \ldots \land (\neg r \vee w \vee q) \]

Their main data structure is a partial variable assignment which represents a solution candidate

\[ V \rightarrow \{t, f\} \]
SAT solving: Deduction

\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

SAT deduces new facts from clauses:

\[ p \mapsto t \]
\[ q \mapsto f \]

At this point, clauses yield no further information
SAT is Abstract Analysis: Deduction

\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

The result of deduction is identical to applying interval analysis to the program:

```c
void foo(void)
{
    bool p, q, r, w;
    if(p)
        if(!p || !q)
            if(q || r || !w)
                if(q || r || w)
                    assert(0);
    p \rightarrow t
    q \rightarrow f
}
```

Deduction in a SAT solver is abstract analysis
SAT solving: Decisions

\[ \phi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

SAT solver makes a “guess”
Pick an unassigned variable and assign a truth value

\[ p \mapsto t \]
\[ q \mapsto f \]

Now new deductions are possible
SAT solving: Learning

\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

\[
\begin{align*}
p & \mapsto t \\
q & \mapsto f \\
r & \mapsto f
\end{align*}
\]

The variable \( w \) would have to be both true and false.

The contradiction is the result of \( r \) being assigned to false as part of a decision. The SAT solver therefore learns that \( r \) must be true:

\[ \varphi \leftarrow \varphi \land r \]

Thursday, 17 January 13
SAT solving: Learning

\[ \varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \]

\[ p \mapsto t \quad \quad p \mapsto t \]
\[ q \mapsto f \quad \quad q \mapsto f \quad \quad \text{conflict} \]
\[ r \mapsto f \quad \quad r \mapsto f \]
\[ w \mapsto f \]

The variable \( w \) would have to be both true and false.

The contradiction is the result of \( r \) being assigned to false as part of a decision. The SAT solver therefore learns that \( r \) must be true:

\[ \varphi \leftarrow \varphi \land r \]
Decisions and learning in a SAT solver are abstract partitioning
SAT is Abstract Analysis

- Deduction in SAT is abstract interpretation
- Decisions and learning are abstract partitioning
- The SAT algorithm is really an automatic partition refinement algorithm.

Domain A

Expanding the scope of SAT
SAT is Abstract Analysis

- Deduction in SAT is abstract interpretation
- Decisions and learning are abstract partitioning
- The SAT algorithm is really an automatic partition refinement algorithm.

Domain A

SAT(A)

Data

Rich logic, e.g. FP

Control

Programs

Prop. Logic

Boolean programs

Expanding the scope of SAT

Thursday, 17 January 13
SAT for programs

**DL0**

\[c_2 : a \leq -1\]
\[c_3 : a \leq 0\]
\[c_3 : a \geq 0\]
\[c_2 : a \geq -1\]
\[c_1 : a \leq -2\]

\[n_2 : b \leq 2\]
\[n_2 : b \geq -2\]

\[\frac{c}{b} : b \leq 0\]
\[\frac{c}{b} : b \geq 0\]

**DL1**

\[n_1 : a \leq -2\]
\[c_1 : \top\]
\[c_2 : \bot\]
\[c_3 : \bot\]
\[c_4 : \bot\]

\[n_2 : b \geq 1\]
\[\frac{c}{b} : \bot\]

\[\neg (n_2 : b \geq 1)\]

SAFE \rightarrow find cut
Prototype: Abstract Conflict Driven Learning (ACDL)

- Implementation over floating-point intervals
- Automatically refines an analysis in a way that is
  - Property dependent
  - Program dependent
- Uses learning to intelligently explore partitions
- Significantly more precise than mature abstract interpreters
- Significantly more efficient than floating-point decision procedures on short non-linear programs
More results

Average speedup over CBMC ~270x
Number of partitions vs. tightness of bound

result $\leq 2.0$

result $\geq -2.0$
Number of partitions vs. tightness of bound

result ≤ 1.5

-\frac{\pi}{2}

\frac{\pi}{2}

result ≥ -1.5
Number of partitions vs. tightness of bound

result $\leq 1.1$

result $\geq -1.1$
Number of partitions vs. tightness of bound

result $\leq 1.01$

result $\geq -1.01$
Number of partitions vs. tightness of bound

\[
\text{result } \leq 1.001
\]

\[
\text{result } \geq -1.001
\]
Current and Future Work

- Develop an SMT solver for floating point logic
- Model on the success of propositional SAT:
  - Simple abstract domain
  - Highly efficient data structures
Current and Future Work

- Develop an SMT solver for floating point logic
- Model on the success of propositional SAT:
  - Simple abstract domain
  - Highly efficient data structures
MathSAT + ACDCL

FP-ACDCL

FP-ACDCL w.o. generalisation

bit-vector encoding (Z3)

(a)

FP-ACDCL w.o. generalisation

(b)
Current and Future Work

- Reengineer prototype into a tool for floating point verification
- Significantly improved efficiency
- Generic interface for integrating abstract domains
- Development and generalisation of heuristics and learning strategies
Current and Future Work

- Reengineer prototype into a tool for floating point verification
- Significantly improved efficiency
- Generic interface for integrating abstract domains
- Development and generalisation of heuristics and learning strategies

Rich logic, e.g. FP

Prop. Logic

Programs

Boolean programs
Conclusion - Part II

Automatic

Scalable
Abstract interpretation
Theorem proving
Decision procedures
Precise

Fully automatic

Scalability
ACDL
Precision
Thank you for your attention