Accurate ICP-based Floating-Point Reasoning

Albert-Ludwigs-Universität Freiburg

Karsten Scheibler, Felix Neubauer, Ahmed Mahdi,
Martin Fränzle, Tino Teige, Tom Bienmüller,
Detlef Fehrer, Bernd Becker

Chair of Computer Architecture

FMCAD 2016

Context of this Work

Context of this Work (1)

Cooperation with Industrypartners (AVACS Transfer Project 1):
“Accurate Dead Code Detection in Embedded C Code by Arithmetic Constraint Solving”

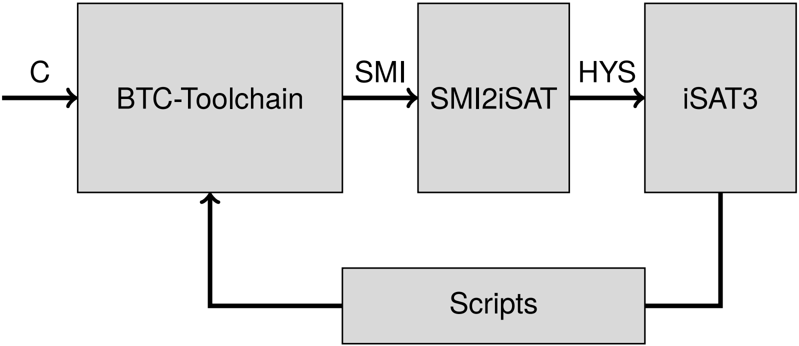
University of Oldenburg:
- Ahmed Mahdi
- Martin Fränzle

University of Freiburg:
- Felix Neubauer
- Karsten Scheibler
- Bernd Becker

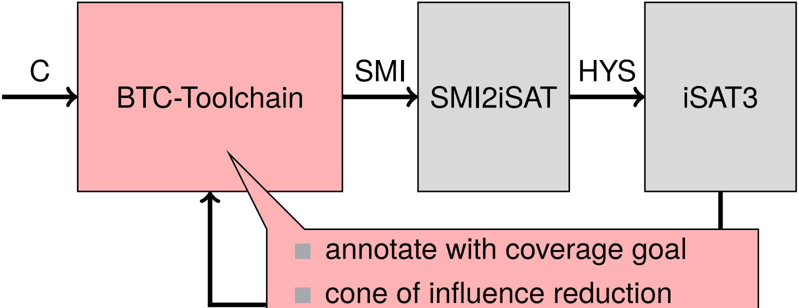
BTC-ES (Oldenburg):
- Tino Teige
- Tom Bienmüller

SICK (Waldkirch):
- Detlef Fehrer

Context of this Work (2)

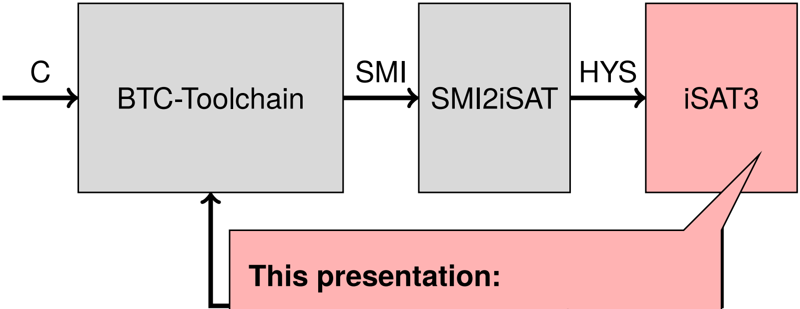


Context of this Work (3)



- annotate with coverage goal
- cone of influence reduction
- resolve loops and functions
- flatten data types
- static single assignment form
- **BMC problem**

Context of this Work (4)



This presentation:

- accurate reasoning for floating-point arithmetic
- support for bitwise integer operations

How does iSAT3 Work

iSAT3 = CDCL + ICP

CDCL: conflict-driven clause learning
ICP: interval constaint propagation

CDCL (1)

CNF

$(\neg b \vee \neg h_1) \wedge$
$(c \vee \neg h_1) \wedge$
$(b \vee \neg c \vee h_1) \wedge$
$(a \vee h_1 \vee \neg h_2) \wedge$
$(a \vee \neg h_1 \vee h_2) \wedge$
$(\neg a \vee h_1 \vee h_2) \wedge$
$(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$
(h_2)

CDCL (1)

CNF		Boolean Formula
$(\neg b \vee \neg h_1) \wedge$ $(c \vee \neg h_1) \wedge$ $(b \vee \neg c \vee h_1) \wedge$ $(a \vee h_1 \vee \neg h_2) \wedge$ $(a \vee \neg h_1 \vee h_2) \wedge$ $(\neg a \vee h_1 \vee h_2) \wedge$ $(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$ (h_2)	⟵ Tseitin-Transformation $(h_1 \Leftrightarrow (\neg b \wedge c))$ $(h_2 \Leftrightarrow (a \oplus h_1))$	$(a \oplus (\neg b \wedge c))$

CDCL (2)

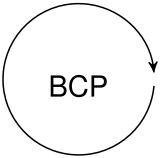
CNF

$(\neg b \vee \neg h_1) \wedge$
$(c \vee \neg h_1) \wedge$
$(b \vee \neg c \vee h_1) \wedge$
$(a \vee h_1 \vee \neg h_2) \wedge$
$(a \vee \neg h_1 \vee h_2) \wedge$
$(\neg a \vee h_1 \vee h_2) \wedge$
$(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$
(h_2)

CDCL (3)

CNF

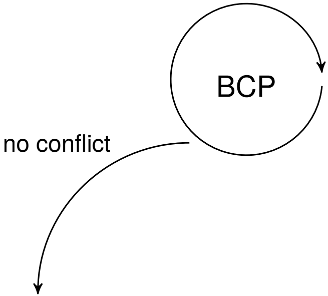
$(\neg b \vee \neg h_1) \wedge$
$(c \vee \neg h_1) \wedge$
$(b \vee \neg c \vee h_1) \wedge$
$(a \vee h_1 \vee \neg h_2) \wedge$
$(a \vee \neg h_1 \vee h_2) \wedge$
$(\neg a \vee h_1 \vee h_2) \wedge$
$(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$
(h_2)



CDCL (4)

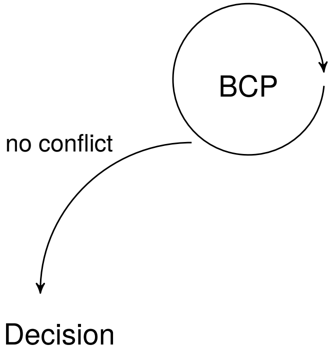
CNF

$(\neg b \vee \neg h_1) \wedge$
$(c \vee \neg h_1) \wedge$
$(b \vee \neg c \vee h_1) \wedge$
$(a \vee h_1 \vee \neg h_2) \wedge$
$(a \vee \neg h_1 \vee h_2) \wedge$
$(\neg a \vee h_1 \vee h_2) \wedge$
$(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$
(h_2)



CNF

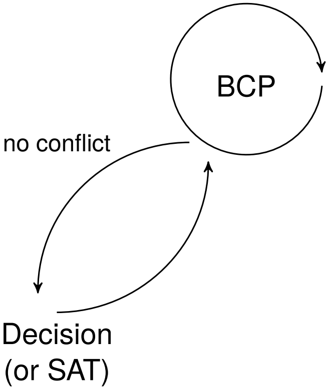
$(\neg b \vee \neg h_1) \wedge$
$(c \vee \neg h_1) \wedge$
$(b \vee \neg c \vee h_1) \wedge$
$(a \vee h_1 \vee \neg h_2) \wedge$
$(a \vee \neg h_1 \vee h_2) \wedge$
$(\neg a \vee h_1 \vee h_2) \wedge$
$(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$
(h_2)



CDCL (4)

CNF

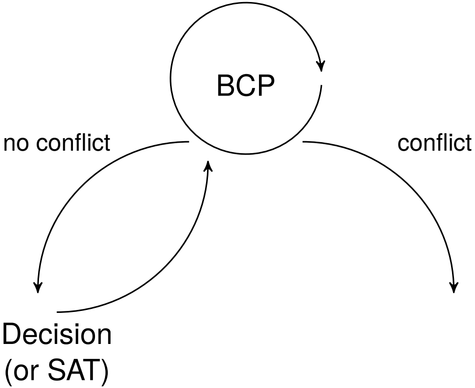
$(\neg b \vee \neg h_1) \wedge$
$(c \vee \neg h_1) \wedge$
$(b \vee \neg c \vee h_1) \wedge$
$(a \vee h_1 \vee \neg h_2) \wedge$
$(a \vee \neg h_1 \vee h_2) \wedge$
$(\neg a \vee h_1 \vee h_2) \wedge$
$(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$
(h_2)



CDCL (5)

CNF

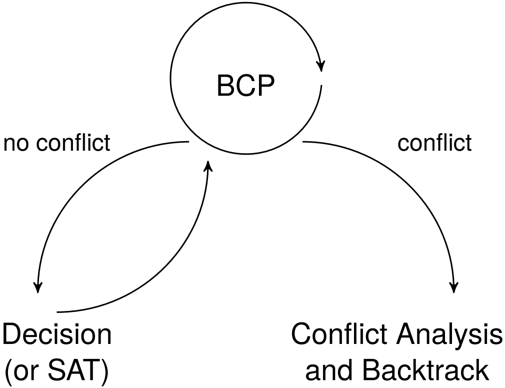
$(\neg b \vee \neg h_1) \wedge$
$(c \vee \neg h_1) \wedge$
$(b \vee \neg c \vee h_1) \wedge$
$(a \vee h_1 \vee \neg h_2) \wedge$
$(a \vee \neg h_1 \vee h_2) \wedge$
$(\neg a \vee h_1 \vee h_2) \wedge$
$(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$
(h_2)



CDCL (5)

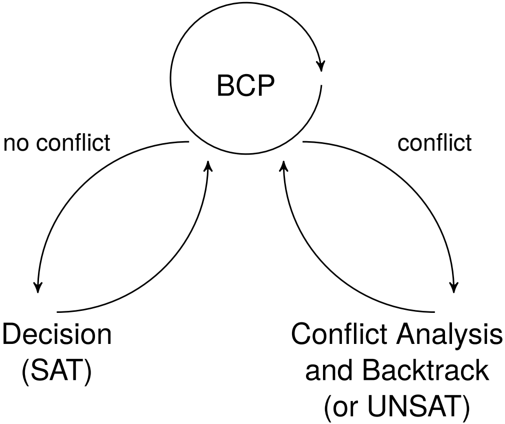
CNF

$(\neg b \vee \neg h_1) \wedge$
$(c \vee \neg h_1) \wedge$
$(b \vee \neg c \vee h_1) \wedge$
$(a \vee h_1 \vee \neg h_2) \wedge$
$(a \vee \neg h_1 \vee h_2) \wedge$
$(\neg a \vee h_1 \vee h_2) \wedge$
$(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$
(h_2)



CNF

$(\neg b \vee \neg h_1) \wedge$
$(c \vee \neg h_1) \wedge$
$(b \vee \neg c \vee h_1) \wedge$
$(a \vee h_1 \vee \neg h_2) \wedge$
$(a \vee \neg h_1 \vee h_2) \wedge$
$(\neg a \vee h_1 \vee h_2) \wedge$
$(\neg a \vee \neg h_1 \vee \neg h_2) \wedge$
(h_2)



iSAT3 (1)

PC + MAP + CNF

$$
\begin{aligned}
&(h_1 = y^2) \land \\
&(h_2 = x + h_1) \land \\
&(h_3 \Leftrightarrow (h_2 < 5)) \land \\
&(a \lor h_3 \lor \neg h_4) \land \\
&(a \lor \neg h_3 \lor h_4) \land \\
&(\neg a \lor h_3 \lor h_4) \land \\
&(\neg a \lor \neg h_3 \lor \neg h_4) \land \\
&(h_4)
\end{aligned}
$$

iSAT3 (1)

PC + MAP + CNF

$$(h_1 = y^2) \wedge$$
$$(h_2 = x + h_1) \wedge$$
$$(h_3 \Leftrightarrow (h_2 < 5)) \wedge$$
$$(a \vee h_3 \vee \neg h_4) \wedge$$
$$(a \vee \neg h_3 \vee h_4) \wedge$$
$$(\neg a \vee h_3 \vee h_4) \wedge$$
$$(\neg a \vee \neg h_3 \vee \neg h_4) \wedge$$
$$(h_4)$$

Tseitin-like Transformation

$$(h_1 = y^2)$$
$$(h_2 = x + h_1)$$
$$(h_3 \Leftrightarrow (h_2 < 5))$$
$$(h_4 \Leftrightarrow (a \oplus h_3))$$

SMT Formula

$$(a \oplus (x + y^2 < 5))$$

linear and nonlinear real arithmetic

with transcendental functions

- maintain interval for every real- or integer-valued variable
- PC: primitive constraints: $(h_1 = y^2)$, $(h_2 = x + h_1)$
- MAP: map literals to simple bounds: $(h_3 \Leftrightarrow (h_2 < 5))$

iSAT3 (1)

PC + [partially hidden]

$(h_1 = y$ …
$(h_2 = x$ …
$(h_3 \Leftrightarrow$ …
$(a \vee h$ …
$(a \vee \neg$ …
$(\neg a \vee$ …
$(\neg a \vee$ …
(h_4)

…ula

…$< 5))$

…al arithmetic

…functions

Assignment

Variable	Type	Value
a	bool	false
x	real	...
y	real	...
h_1	real	...
h_2	real	h_3
h_3	bool simple bound $(h_2 < 5)$	true
h_4	bool	true

- maint… …able
- PC: p…
- MAP: map literals to simple bounds: $(h_3 \Leftrightarrow (h_2 < 5))$

iSAT3 (2)

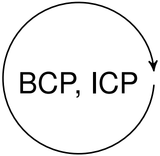
PC + MAP + CNF

$$
\begin{aligned}
&(h_1 = y^2) \wedge \\
&(h_2 = x + h_1) \wedge \\
&(h_3 \Leftrightarrow (h_2 < 5)) \wedge \\
&(a \vee h_3 \vee \neg h_4) \wedge \\
&(a \vee \neg h_3 \vee h_4) \wedge \\
&(\neg a \vee h_3 \vee h_4) \wedge \\
&(\neg a \vee \neg h_3 \vee \neg h_4) \wedge \\
&(h_4)
\end{aligned}
$$

iSAT3 (3)

PC + MAP + CNF

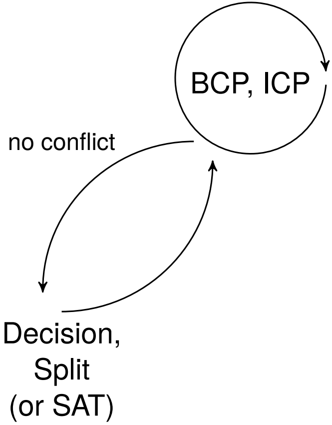
$(h_1 = y^2) \wedge$
$(h_2 = x + h_1) \wedge$
$(h_3 \Leftrightarrow (h_2 < 5)) \wedge$
$(a \vee h_3 \vee \neg h_4) \wedge$
$(a \vee \neg h_3 \vee h_4) \wedge$
$(\neg a \vee h_3 \vee h_4) \wedge$
$(\neg a \vee \neg h_3 \vee \neg h_4) \wedge$
(h_4)



iSAT3 (4)

PC + MAP + CNF

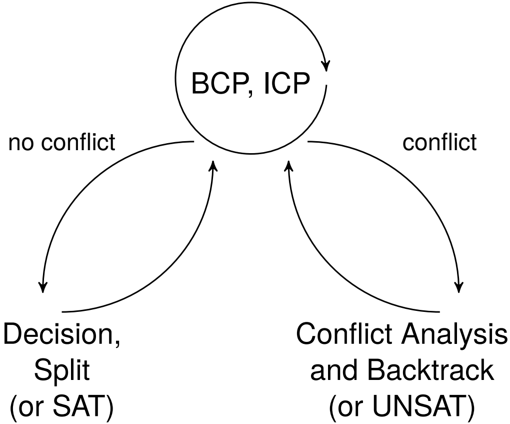
$$
\begin{aligned}
&(h_1 = y^2) \wedge \\
&(h_2 = x + h_1) \wedge \\
&(h_3 \Leftrightarrow (h_2 < 5)) \wedge \\
&(a \vee h_3 \vee \neg h_4) \wedge \\
&(a \vee \neg h_3 \vee h_4) \wedge \\
&(\neg a \vee h_3 \vee h_4) \wedge \\
&(\neg a \vee \neg h_3 \vee \neg h_4) \wedge \\
&(h_4)
\end{aligned}
$$



iSAT3 (5)

PC + MAP + CNF

$(h_1 = y^2) \wedge$
$(h_2 = x + h_1) \wedge$
$(h_3 \Leftrightarrow (h_2 < 5)) \wedge$
$(a \vee h_3 \vee \neg h_4) \wedge$
$(a \vee \neg h_3 \vee h_4) \wedge$
$(\neg a \vee h_3 \vee h_4) \wedge$
$(\neg a \vee \neg h_3 \vee \neg h_4) \wedge$
(h_4)



iSAT3 (6)

	SAT	iSAT3
Deductions	• BCP for clauses	• BCP for clauses evaluate simple bound literals $\rightsquigarrow$ implication clauses • ICP for PC $\rightsquigarrow$ arithmetic clauses
Decisions	• decide literals	• decide literals • generate new simple bound literals and decide them
Conflict Analyses	• traverse implication graph (1UIP) $\rightsquigarrow$ conflict clauses	• traverse implication graph (1UIP) $\rightsquigarrow$ conflict clauses

iSAT3 (6)

	SAT	iSAT3
Deductions	• BCP for clauses	• BCP for clauses evaluate simple bound literals ⇝ **implication clauses** • ICP for PC ⇝ **arithmetic clauses**
Decisions	• decide literals	• decide literals • generate new simple bound literals and decide them
Conflict Analyses	• traverse implication graph (1UIP) ⇝ conflict clauses	• traverse implication graph (1UIP) ⇝ conflict clauses

iSAT3 (7)

Implication Clauses:

- unassigned simple bound literals are evaluated lazily
- therefore implications possible: $(h_2 < 5) \Rightarrow (h_2 < 7)$

Arithmetic Clauses:

- result of interval constraint propagation (ICP)
- e.g. $h_2 = x + h_1$: $((x \leq 3) \wedge (h_1 < 2)) \Rightarrow (h_2 < 5)$
- redirect, e.g. $x = h_2 - h_1$: $((h_2 < 10) \wedge (h_1 \geq 1)) \Rightarrow (x < 9)$
- using floating-point numbers for interval bounds
- always round outwards for safe enclosing intervals
- generate new simple bound literals

iSAT3 Summarized

iSAT3 = CDCL + ICP, **goes beyond CDCL(T)**:

Boolean abstraction contains	
CDCL(T)	iSAT3
combinations of truth values of the theory atoms	interval bounds of theory variables and sub-expressions

iSAT3 Summarized

iSAT3 = CDCL + ICP, **goes beyond CDCL(T)**:

Boolean abstraction contains	
CDCL(T)	iSAT3
combinations of truth values of the theory atoms	interval bounds of theory variables and sub-ex ssions

iSAT3 is the 3rd implementation of the iSAT algorithm. Abstract CDCL with interval abstraction has similarities to the iSAT algorithm

iSAT algorithm: *"Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure"*, JSAT 2007

Abstract CDCL: *"Deciding Floating-Point Logic with Systematic Abstraction"*, FMCAD 2012

iSAT3 Summarized

iSAT3 = CDCL + ICP, **goes beyond CDCL(T)**:

Boolean abstraction contains	
CDCL(T)	iSAT3
combinations of truth values of the theory atoms	interval bounds of theory variables and sub-expressions

1. **new arithmetic operations $\rightsquigarrow$ add ICP-contractors**
2. **need to adapt Boolean abstraction for floating-point**

Accurate Reasoning for Floating-Point Arithmetic

Accurate Reasoning for FP (1)

UNI FREIBURG

IEEE-754 Specification (float, 32 bits)

Bitpos $\rightarrow$ 31	30 ... 23	22 ... 0
sign	exponent	fraction / mantissa

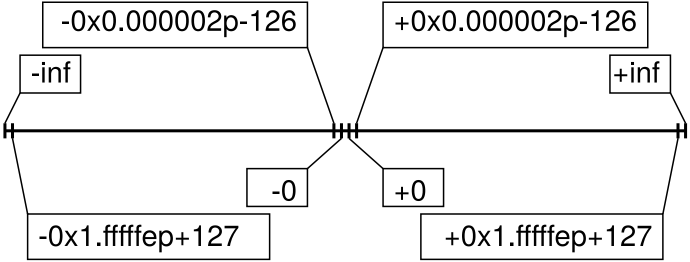
1. normal numbers:
   - mantissa bitpos 23 assumed to be 1
   - exponent $1 \rightsquigarrow -126 \quad \dots \quad 254 \rightsquigarrow +127$
   - sign $0 \rightsquigarrow$ positive $\quad 1 \rightsquigarrow$ negative
2. special numbers:
   - signed zeros (-0, $+0$)
   - $-\infty, +\infty$ (-inf, +inf)
   - subnormal numbers (leading zeros in mantissa)
   - not a number (NaN)
3. rounding modes (up, down, **nearest**)

Accurate Reasoning for FP (2)

32 bit floating-point values and their ordering

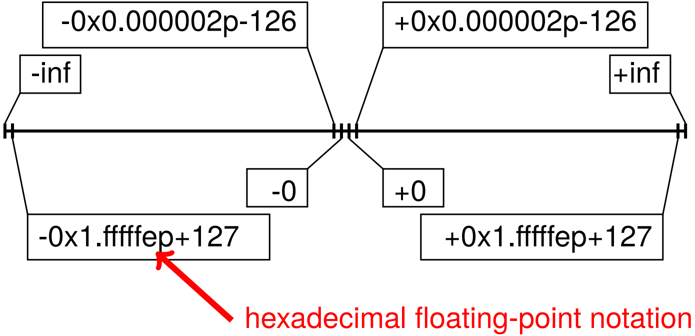
Accurate Reasoning for FP (2)

32 bit floating-point values and their ordering



Accurate Reasoning for FP (2)

32 bit floating-point values and their ordering



Accurate Reasoning for FP (3)

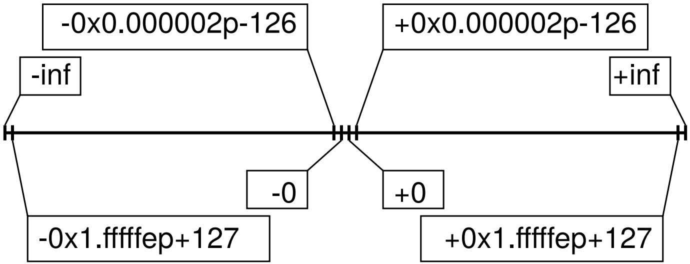
simple bound ordering:

-inf $<$ -0x1.fffffep+127 $< \ldots$
$\ldots <$ -0x0.000002p-126 $<$ -0 $<$ +0 $<$ +0x0.000002p-126 $< \ldots$
$\ldots <$ +0x1.fffffep+127 $<$ +inf

- no strict bounds needed:
  reals: $(x \leq 5) \Leftrightarrow \neg(x > 5)$
  floating-point: $(x \leq$-0x0.000002p-126$) \Leftrightarrow \neg(x \geq$-0$)$
- floating-point comparison operators and signed zeros:
  - `(x <= 0)` $\rightsquigarrow (x \leq +0)$
  - `(x >= 0)` $\rightsquigarrow (x \geq -0)$
  - `(x == 0)` $\rightsquigarrow (x \geq -0) \wedge (x \leq +0)$

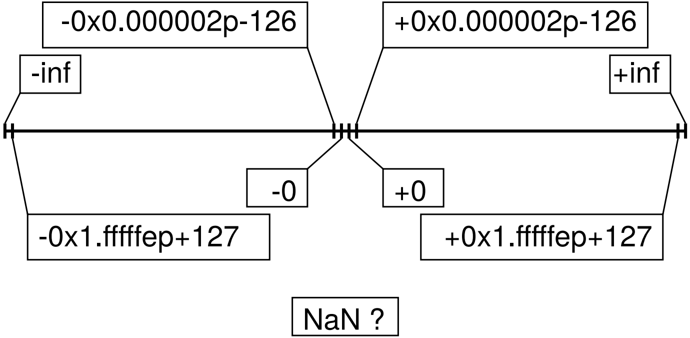
Accurate Reasoning for FP (3)

32 bit floating-point values and their ordering



Accurate Reasoning for FP (3)

32 bit floating-point values and their ordering



Accurate Reasoning for FP (4)

```
#include <math.h>
#include <stdio.h>
int main(void) {
    double a = sqrt(-1);
    printf("%1.2f\n", a);

    if (a < 0) printf("if\n");
    else printf("else\n");

    if (a >= 0) printf("if\n");
    else printf("else\n");
    return (0);
    }
```

```
-nan
else
else
```

Accurate Reasoning for FP (4)

```
#include <math.h>
#include <stdio.h>
int main(void) {
    double a = sqrt(-1);
    printf("%1.2f\n", a);

    if (a <= 0) printf("if\n");
    else printf("else\n");

    if (a > 0) printf("if\n");
    else printf("else\n");
    return (0);
    }
```

```
-nan
else
else
```

Accurate Reasoning for FP (4)

```
#include <math.h>
#include <stdio.h>
int main(void) {
    double a = sqrt(-1);
    printf("%1.2f\n", a);

    if (a == 0) printf("if\n");
    else printf("else\n");

    if (a != 0) printf("if\n");
    else printf("else\n");
    return (0);
    }
```

```
-nan
else
if
```

Accurate Reasoning for FP (5)

	SAT	iSAT3
Deductions	• BCP for clauses	• BCP for clauses evaluate simple bound literals $\rightsquigarrow$ **implication clauses** • ICP for PC $\rightsquigarrow$ **arithmetic clauses**
Decisions	• decide literals	• decide literals • generate new simple bound literals and decide them
Conflict Analyses	• traverse implication graph (1UIP) $\rightsquigarrow$ conflict clauses	• traverse implication graph (1UIP) $\rightsquigarrow$ conflict clauses

Accurate Reasoning for FP (5)

- NaN incomparable against all other values:
  $(x \sim NaN)$, $\sim \in \{<, \leq, =, \geq, >\}$ is always false

- adapt Boolean encoding: special literal x_{NaN}

x_{NaN}	x is NaN
$\neg x_{NaN}$	x is determined by simple bound literals $(x \leq$-inf$) \ldots (x \leq$-0$) \ldots$

Accurate Reasoning for FP (5)

- implication clauses:
  $(\neg x_{NaN} \wedge (x \leq 5)) \Rightarrow (x \leq 7)$

- arithmetic clauses: $h = x + y$
  $(\neg x_{NaN} \wedge \neg y_{NaN} \wedge \neg h_{NaN} \wedge (x \leq 3) \wedge (y \leq 2)) \Rightarrow (h \leq 5)$

- not shown here, but x_{NaN} also relevant during Tseitin-like transformation

- besides $<, \leq, =, \geq, >$ operators, further operator to mimic behaviour of assignments: `x = y` vs. `x == y`

Accurate Reasoning for FP (6)

New ICP-Contractors for $+, -, *, /$ (round-to-nearest):

1. NaN cases: handled outside with separate clauses
2. forward deduction: execute operation with round-to-nearest
3. backward deduction: only redirecting the primitive constraint is not enough

Accurate Reasoning for FP (6)

New ICP-Contractors for $+, -, *, /$ (round-to-nearest):

1. NaN cases: handled outside with separate clauses
2. forward deduction: execute operation with round-to-nearest
3. backward deduction: only redirecting the primitive constraint is not enough

ICP-contractors called when NaN-literals of operands false (otherwise the created arithmetic clauses not unit)

Accurate Reasoning for FP (6)

1. Separate clauses for primitive constraint ($h = x + y$):

x or y is NaN	$\Rightarrow$	h is NaN
x and y are infinities with opposite signs	$\Rightarrow$	h is NaN
x and y are not NaN and x is never -inf or +inf	$\Rightarrow$	h is not NaN
x and y are not NaN and y is never -inf or +inf	$\Rightarrow$	h is not NaN
x and y are not NaN and x and y are never -inf	$\Rightarrow$	h is not NaN
x and y are not NaN and x and y are never +inf	$\Rightarrow$	h is not NaN

$$(\neg x_{NaN} \vee h_{NaN}) \wedge (\neg y_{NaN} \vee h_{NaN}) \wedge$$
$$(x_{NaN} \vee y_{NaN} \vee \neg(x \leq \text{-inf}) \vee \neg(y \geq \text{+inf}) \vee h_{NaN}) \wedge$$
$$(x_{NaN} \vee y_{NaN} \vee \neg(x \geq \text{+inf}) \vee \neg(y \leq \text{-inf}) \vee h_{NaN}) \wedge$$
$$(x_{NaN} \vee y_{NaN} \vee (x \leq \text{-inf}) \vee (x \geq \text{+inf}) \vee \neg h_{NaN}) \wedge$$
$$(x_{NaN} \vee y_{NaN} \vee (y \leq \text{-inf}) \vee (y \geq \text{+inf}) \vee \neg h_{NaN}) \wedge$$
$$(x_{NaN} \vee y_{NaN} \vee (x \leq \text{-inf}) \vee (y \leq \text{-inf}) \vee \neg h_{NaN}) \wedge$$
$$(x_{NaN} \vee y_{NaN} \vee (x \geq \text{+inf}) \vee (y \geq \text{+inf}) \vee \neg h_{NaN})$$

Accurate Reasoning for FP (6)

2 Forward deduction primitive constraint ($h = x + y$):

$h \in$ [-inf, +inf],
$x \in$ [0x1.1p+100, 0x1.1p+100],
$y \in$ [-0x1.1p+11, -0x1.1p+10]

$h_{lb} = x_{lb} + y_{lb}$ =0x1.1p+100 + -0x1.1p+11=0x1.1p+100
$h_{ub} = x_{ub} + y_{ub}$ =0x1.1p+100 + -0x1.1p+10=0x1.1p+100

apply operation with round-to-nearest

Accurate Reasoning for FP (6)

3 Backward deduction primitive constraint ($h = x + y$):

$h \in$ [0x1.1p+100, 0x1.1p+100],
$x \in$ [0x1.1p+100, 0x1.1p+100],
$y \in$ [-0x1.1p+11, -0x1.1p+10]

$$\begin{array}{lllllll} y_{lb} & = & h_{lb} - x_{ub} & = & \text{0x1.1p+100} - \text{0x1.1p+100} & = & 0 \\ y_{ub} & = & h_{ub} - x_{lb} & = & \text{0x1.1p+100} - \text{0x1.1p+100} & = & 0 \end{array}$$

[-0x1.1p+11, -0x1.1p+10] $\cap [0,0] = \emptyset$
simply redirecting and rounding outward is **WRONG!**

Accurate Reasoning for FP (6)

3 Backward deduction primitive constraint ($h = x + y$):

$h \in$ [0x1.1p+100, 0x1.1p+100],
$x \in$ [0x1.1p+100, 0x1.1p+100],
$y \in$ [-0x1.1p+11, -0x1.1p+10]

$$
\begin{array}{lllll}
y_{lb} & = & h_{lb} - x_{ub} & = & \mathit{prev}(\texttt{0x1.1p+100}) - \mathit{next}(\texttt{0x1.1p+100}) \\
 & & & = & \texttt{0x1.0ffffep+100 - 0x1.100002p+100} \\
 & & & = & \texttt{-0x1.000000p+78} \\
y_{ub} & = & h_{ub} - x_{lb} & = & \mathit{next}(\texttt{0x1.1p+100}) - \mathit{prev}(\texttt{0x1.1p+100}) \\
 & & & = & \texttt{0x1.100002p+100 - 0x1.0ffffep+100} \\
 & & & = & \texttt{0x1.000000p+78}
\end{array}
$$

Accurate Reasoning for FP Summarized

- floating-point arithmetic contains special values
- ordering possible, except NaN
- unordered NaN $\rightsquigarrow$ adapted Boolean encoding
  - implication clauses
  - arithmetic clauses
- new ICP-contractors for floating-point operations
  - NaN-cases handled with BCP
  - outward rounding not enough in backward deduction

ICP-Contractors for Bitwise Integer Operations

ICP-Contractors for Bitwise Operations (1)

- operating on intervals
- a bit-pattern can be interpreted as signed or unsigned

	0001 0001	1000 0001
`signed char`	17	-127
`unsigned char`	17	129

- need to know bitwidth and signedness of each operation
- S_NOT(*arg*,*bitwidth*), U_NOT(*arg*,*bitwidth*)
- S_AND(*arg*1,*arg*2,*bitwidth*), U_AND(*arg*1,*arg*2,*bitwidth*)
- S_OR(*arg*1,*arg*2,*bitwidth*), U_OR(*arg*1,*arg*2,*bitwidth*)
- S_XOR(*arg*1,*arg*2,*bitwidth*), U_XOR(*arg*1,*arg*2,*bitwidth*)
- S_CAST(*arg*,*bitwidth*), U_CAST(*arg*,*bitwidth*)

ICP-Contractors for Bitwise Operations (2)

- $(h = x + y)$, $x \in [1,7]$, $y \in [1,8]$:
  $$\begin{array}{lllll} h_{lb} & = & x_{lb} + y_{lb} & = & 1+1 & = & 2 \\ h_{ub} & = & x_{ub} + y_{ub} & = & 7+8 & = & 15 \end{array}$$
  $\rightsquigarrow$ operating on bounds **OK**

- $(h = \text{U\_AND}(x,y,8))$, $x \in [1,7]$, $y \in [1,8]$:
  $$\begin{array}{lllllll} h_{lb} & = & x_{lb} \,\&\, y_{lb} & = & 1 \,\&\, 1 & = & 1 & \quad (1 \,\&\, 2 = 0) \\ h_{ub} & = & x_{ub} \,\&\, y_{ub} & = & 7 \,\&\, 8 & = & 0 & \quad (7 \,\&\, 7 = 7) \end{array}$$
  $\rightsquigarrow$ operating on bounds **WRONG**

ICP-Contractors for Bitwise Operations (3)

UNI FREIBURG

1. use addition, subtraction, minimum and maximum to get safe overapproximations of the lower and upper bounds,
   e.g. ($h = \text{U\_AND}(x,y,8)$), $x \in [1,7]$, $y \in [1,8]$:

   $$h_{ub} \quad = \quad min(x_{ub}, y_{ub}) \quad = \quad min(7,8) \quad = \quad 7$$

2. exploit common bit-prefixes,
   e.g. ($h = \text{U\_AND}(x,y,8)$), $x \in [18,30]$, $y \in [89,92]$:

   $$\begin{array}{lllll} x_{lb} & = & 18 & = & 0001\,0010 \\ x_{ub} & = & 30 & = & 0001\,1110 \\ & & & & 0001 \end{array} \qquad \text{common bit-prefix for values in } x$$

   $$\begin{array}{lllll} y_{lb} & = & 89 & = & 0101\,1001 \\ y_{ub} & = & 92 & = & 0101\,1100 \\ & & & & 0101\,1 \end{array} \qquad \text{common bit-prefix for values in } y$$

   $$\begin{array}{lll} h_{lb} & = & 0001\,0000 \;\&\; 0101\,1000 \;=\; 0001\,0000 \;=\; 16 \qquad \text{trailing bits are 0} \\ h_{ub} & = & 0001\,1111 \;\&\; 0101\,1111 \;=\; 0001\,1111 \;=\; 31 \qquad \text{trailing bits are 1} \end{array}$$

ICP-Contractors for Bitwise Operations (3)

1. use addition, subtraction, minimum and maximum to get safe overapproximations of the lower and upper bounds,
   e.g. ($h = \text{U\_AND}(x, y, 8)$), $x \in [1,7]$, $y \in [1,8]$:

A detailed description of all operations can be found in AVACS Technical Report 116:
"Extending iSAT3 with ICP-Contractors for Bitwise Integer Operations"

$$
\begin{array}{lcccl}
y_{lb} & = & 89 & = & 01011001 \\
y_{ub} & = & 92 & = & 01011100 \\
 & & & & 01011
\end{array}
$$

common bit-prefix for values in y

$$
\begin{array}{lclcl}
h_{lb} & = & 00010000 \,\&\, 01011000 = 00010000 = 16 & & \text{trailing bits are 0} \\
h_{ub} & = & 00011111 \,\&\, 01011111 = 00011111 = 31 & & \text{trailing bits are 1}
\end{array}
$$

Optimizations

Intermediate Point-Splits

- decomposition into PCs might lead to coarser intervals, e.g. $((x+y)-x \leq 7) \rightsquigarrow (h_1 = x+y) \wedge (h_2 = h_1 - x)$
  $x, y \in [0,10] : h_1 \in [0,20], h_2 \in [-10,30] \supset [0,10]$

- tighter intervals if x is point interval

- change decision heuristic, every k-th interval split will assign a point interval ($k = 4$)

- might help to find a solution, **BUT:** detrimental for conflict clauses

Global-ICP (1)

$$\ldots \wedge (a \rightarrow (i_1 - i_2 = 0)) \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

Global-ICP (1)

$$\ldots \wedge (a \rightarrow (i_1 - i_2 = 0)) \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

with $a = 1$:

$$\ldots \wedge (i_1 - i_2 = 0) \quad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

$$\ldots \wedge (i_1 = i_2) \quad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

$$\ldots \quad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_1, 0), 32)) \wedge \ldots$$

Global-ICP (1)

$$\ldots \wedge (a \rightarrow (i_1 - i_2 = 0)) \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

with $a = 1$:

$$\ldots \wedge (i_1 - i_2 = 0) \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

$$\ldots \wedge (i_1 = i_2) \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

$$\ldots \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_1, 0), 32)) \wedge \ldots$$

with $b = 1$:

$$\ldots \wedge (i_1 \neq \text{S\_CAST}(i_1, 32)) \wedge \ldots$$

Global-ICP (1)

$$\ldots \wedge (a \rightarrow (i_1 - i_2 = 0)) \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

with $a = 1$:

$$\ldots \wedge (i_1 - i_2 = 0) \quad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

$$\ldots \wedge (i_1 = i_2) \quad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

$$\ldots \quad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_1, 0), 32)) \wedge \ldots$$

with $b = 1$:

$$\ldots \quad \wedge (i_1 \neq \text{S\_CAST}(i_1, 32)) \wedge \ldots$$

with $i_1 \in [0, 2^{31} - 1]$:

$$\ldots \quad \wedge (i_1 \neq i_1) \wedge \ldots$$

Global-ICP (1)

$$\ldots \wedge (a \rightarrow (i_1 - i_2 = 0)) \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

with $a = 1$:

$$\ldots \wedge (i_1 - i_2 = 0) \qquad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$
$$\ldots \wedge (i_1 = i_2) \qquad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$
$$\ldots \qquad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_1, 0), 32)) \wedge \ldots$$

with $b = 1$:

$$\ldots \qquad \wedge (i_1 \neq \text{S\_CAST}(i_1, 32)) \wedge \ldots$$

with $i_1 \in [0, 2^{31} - 1]$:

$$\ldots \qquad \wedge (i_1 \neq i_1) \wedge \ldots$$

but this symbolic dependency is not visible for ICP

$(h_1 = i_1 - i_2) \wedge$
$(h_2 = \text{ITE}(b, i_2, 0)) \wedge$
$(h_3 = \text{S\_SCAST}(h_2, 32)) \wedge$
$(h_4 = i_1 - h_3)$

just looking at these primitive constraints

Global-ICP (1)

$$\ldots \wedge (a \rightarrow (i_1 - i_2 = 0)) \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

with $a = 1$:

$$\ldots \wedge (i_1 - i_2 = 0) \qquad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

$$\ldots \wedge (i_1 = i_2) \qquad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_2, 0), 32)) \wedge \ldots$$

$$\ldots \qquad \wedge (i_1 \neq \text{S\_CAST}(\text{ITE}(b, i_1, 0), 32)) \wedge \ldots$$

with $b = 1$:

$$\ldots \qquad \wedge (i_1 \neq \text{S\_CAST}(i_1, 32)) \wedge \ldots$$

with $i_1 \in [0, 2^{31} - 1]$:

$$\ldots \qquad \wedge (i_1 \neq i_1) \wedge \ldots$$

but this symbolic dependency is not visible for ICP

$(h_1 = i_1 - i_2) \wedge$
$(h_2 = \text{ITE}(b, i_2, 0)) \wedge$
$(h_3 = \text{S\_SCAST}(h_2, 32)) \wedge$
$(h_4 = i_1 - h_3)$

just looking at these primitive constraints

ICP with smallest possible bound improvement for i_1:

$$\rightsquigarrow [1, 2^{31} - 1] \rightsquigarrow [2, 2^{31} - 1] \rightsquigarrow [2, 2^{31} - 2] \rightsquigarrow \ldots$$

Global-ICP (2)

- ICP with smallest possible bound improvement for i_1:
  $\rightsquigarrow [1, 2^{31} - 1] \rightsquigarrow [2, 2^{31} - 1] \rightsquigarrow [2, 2^{31} - 2] \rightsquigarrow \ldots$
- more than 64 deductions per variable per decision level:
  1. no further deductions for this variable
  2. analyze implication graph, collect involved primitive constraints (the 4 PCs from previous slide)
- analyze primitive constraints semi-symbolically
- conflicting clause which spans more than one PC, e.g.
  $(b \wedge (h_1 \geq 0) \wedge (h_1 \leq 0) \wedge (i_2 \geq 0) \wedge (i_2 \leq 2^{31} - 1)) \Rightarrow (h_4 \leq 0)$

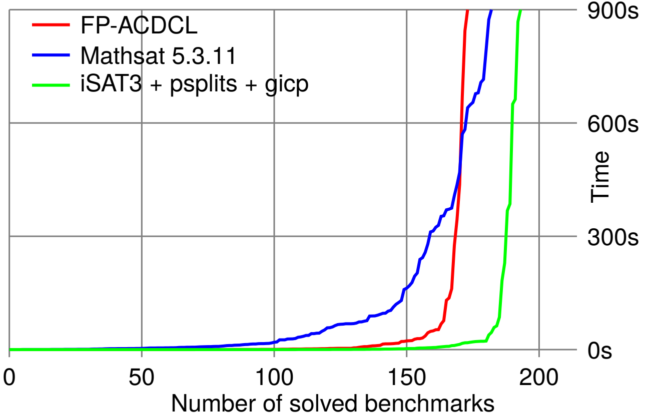
Results

Results (1)

- 213 **pure floating-point** benchmarks from the FP-ACDCL paper
- Comparison between FP-ACDCL (ICP-based), Mathsat (bit-blasting) and iSAT3 (ICP-based)
- Timeout: 900 seconds, Memout: 2 GB

Solver	S+U	SAT	UNSAT	TO	MO
FP-ACDCL	173	97	76	40	0
Mathsat 5.3.11	182	101	81	23	8
iSAT3	164	90	74	47	2
iSAT3 + psplits	186	111	75	27	0
iSAT3 + psplits + gicp	193	111	82	20	0

Results (1)



FP-ACDCL
Mathsat 5.3.11
iSAT3 + psplits + gicp
900s
600s
300s
0s
Time
0
50
100
150
200
Number of solved benchmarks

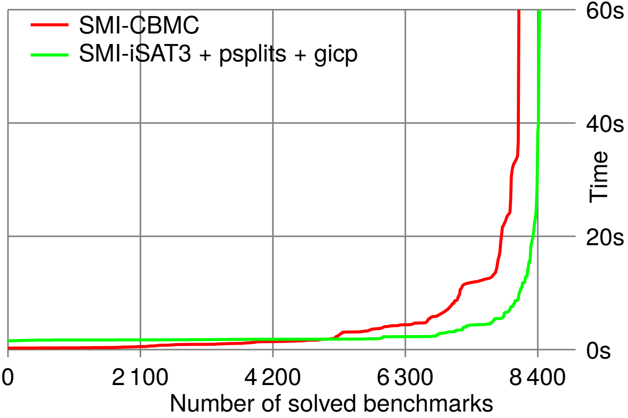
Results (2)

- 8778 BMC benchmarks generated by BTC toolchain, containing **floating-point and bitwise integer operations**
- Comparison between CBMC (bit-blasting, k-induction) and iSAT3 (ICP-based, Craig interpolation)

  both with on-the-fly translation from SMI to their input language
- Timeout: 60 seconds

Solver	S+U	SAT	U51	U∞	TO
SMI-CBMC	8099	7424	44	631	679
SMI-iSAT3	7647	6671	153	823	1131
SMI-iSAT3 + psplits	8169	7192	156	821	609
SMI-iSAT3 + psplits + gicp	8430	7427	172	831	348

Results (2)


SMI-CBMC
SMI-iSAT3 + psplits + gicp
60s
40s
20s
0s
Time
0
2 100
4 200
6 300
8 400
Number of solved benchmarks

Conclusion

Conclusion

- dead-code detection in C programs = accurate floating-point reasoning + bitwise integer operations
- iSAT3: first non-bit-blasting SMT solver supporting the full range of basic data types and operations in C programs
- promising results:
  - outperforms bit-blasting solvers (MathSAT, CBMC)
  - outperforms other ICP-based solver (FP-ACDCL)
- Outlook: also integrate ICP-contractors for floating-point sine, cosine