Satisfiability Modulo Theories

Using Open-Source to solve hard problems

Gereon Kremer
FrOSCon, August 5th, 2023
What?

**libsolv**

Package dependency solver using a satisfiability algorithm

`libsolv` is a library for solving packages and reading repositories. The solver uses a satisfiability algorithm.
libsolv

Package dependency solver using a satisfiability algorithm

libsolv is a library for solving packages and reading repositories. The solver uses a satisfiability algorithm.

CBMC is a Bounded Model Checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio. A variant of CBMC that analyses Java bytecode is available as JBLMC.

CBMC verifies memory safety (which includes array bounds checks and checks for the safe use of pointers), checks for exceptions, checks for various variants of undefined behavior, and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.
libsolv

Package dependency solver using a satisfiability algorithm

libsolv is a library for solving packages and reading repositories. The solver uses a satisfiability algorithm.

VCC

VCC is a tool that proves correctness of annotated concurrent C programs or finds problems in them. VCC extends C with design by contract features, like pre- and postcondition as well as type invariants. Annotated programs are translated to logical formulas using the Boogie tool, which passes them to an automated SMT solver Z3 to check their validity.

CBMC

CBMC is a Bounded Model Checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio. A variant of CBMC that analyses Java bytecode is available as JBCMC.

CBMC verifies memory safety (which includes array bounds checks and checks for the safe use of pointers), checks for exceptions, checks for various variants of undefined behavior, and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.
**libolv**

Package dependency solver using a satisfiability algorithm

libolv is a library for solving packages and reading repositories. The solver uses a satisfiability algorithm.

CBMC is a Bounded Model Checker for C and C++ programs. It supports C99, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio. A variant of CBMC that analyses Java bytecode is available as JBMC.

CBMC verifies memory safety (which includes array bounds checks and checks for the safe use of pointers), checks for exceptions, checks for various variants of undefined behavior, and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.

**LifeJacket: Verifying precise floating-point optimizations in LLVM**

Andros Nötzli, Fraser Brown

Optimizing floating-point arithmetic is vital because it is ubiquitous, costly, and used in compute-heavy workloads. Implementing precise optimizations correctly, however, is difficult, since developers must account for all the esoteric properties of floating-point arithmetic to ensure that their transformations do not alter the output of a program. Manual reasoning is error prone and stifles incorporation of new optimizations. We present an approach to automate reasoning about floating-point optimizations using satisfiability modulo theories (SMT) solvers. We implement the approach in LifeJacket, a system for automatically verifying precise floating-point optimizations for the LLVM assembly language. We have used LifeJacket to verify 43 LLVM optimizations and to discover eight incorrect ones, including three previously unreported problems. LifeJacket is an open source extension of the Alive system for optimization verification.
What?

**libsov**

Package dependency solver using a satisfiability algorithm

libsov is a library for solving packages and reading repositories. The solver uses a satisfiability algorithm.

**KLEE Symbolic Execution Engine**

KLEE is a dynamic symbolic execution engine built on top of the LLVM compiler infrastructure, and available under the UJIC open source license. For more information on what KLEE is and what it can do, see the 2006 paper.

**VCC**

VCC is a tool that proves correctness of annotated concurrent C programs or finds problems in them. VCC extends C with design by contract features, like pre- and postcondition as well as type invariants. Annotated programs are translated to logical formulas using the Boogie tool, which passes them to an automated SMT solver Z3 to check their validity.

CBMC is a Bounded Model Checker for C and C++ programs. It supports C99, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio. A variant of CBMC that analyses Java bytecode is available as JBC.

CBMC verifies memory safety (which includes array bounds checks and checks for the safe use of pointers), checks for exceptions, checks for various variants of undefined behavior, and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.

**LifeJacket: Verifying precise floating-point optimizations in LLVM**

Andreas Nützi, Fraser Brown

Optimizing floating-point arithmetic is vital because it is ubiquitous, costly, and used in compute-heavy workloads. Implementing precise optimizations correctly, however, is difficult, since developers must account for all the exotic properties of floating-point arithmetic to ensure that their transformations do not alter the output of a program. Manual reasoning is error prone and stifles incorporation of new optimizations. We present an approach to automate reasoning about floating-point optimizations using satisfiability modulo theories (SMT) solvers. We implement the approach in LifeJacket, a system for automatically verifying precise floating-point optimizations for the LLVM assembly language. We have used LifeJacket to verify 43 LLVM optimizations and to discover eight incorrect ones, including three previously unreported problems. LifeJacket is an open source extension of the Alive system for optimization verification.
What?

**libsvolv**

Package dependency solver using a satisfiability algorithm

libsvolv is a library for solving packages and reading repositories. The solver uses a satisfiability algorithm.

**VCC**

VCC is a tool that proves correctness of annotated concurrent C programs or finds problems in them. VCC extends C with design by contract features, like pre- and postcondition as well as type invariants. Annotated programs are translated to logical formulas using the Boogie tool, which passes them to an automated SMT solver Z3 to check their validity.

**CBMC**

CBMC is a Bounded Model Checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio. A variant of CBMC that analyses Java bytecode is available as JBMC.

CBMC verifies memory safety (which includes array bounds checks and checks for the safe use of pointers), checks for exceptions, checks for various variants of undefined behavior, and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.

**LifeJacket: Verifying precise floating-point optimizations in LLVM**

Andros Nötzig, Fraser Brown

Optimizing floating-point arithmetic is vital because it is ubiquitous, costly, and used in compute-heavy workloads. Implementing precise optimizations correctly, however, is difficult, since developers must account for all the esoteric properties of floating-point arithmetic to ensure that their transformations do not alter the output of a program. Manual reasoning is error prone and stifies incorporation of new optimizations. We present an approach to automate reasoning about floating-point optimizations using satisfiability modulo theories (SMT) solvers. We implement the approach in LifeJacket, a system for automatically verifying precise floating-point optimizations for the LLVM assembly language. We have used LifeJacket to verify 43 LLVM optimizations and to discover eight incorrect ones, including three previously unreported problems. LifeJacket is an open-source extension of the Alvis system for optimization verification.

**A billion SMT queries a day**

Neha Rungta | August 18, 2022

CAV keynote lecture by the director of applied science for AWS Identity explains how AWS is making the power of automated reasoning available to all customers.

**KLEE Symbolic Execution Engine**

KLEE is a dynamic symbolic execution engine built on top of the LLVM compiler infrastructure and available under the University of Illinois open source license. For more information on what KLEE is and what it can do, see the [KLEE paper](https://arxiv.org/abs/1206.2007).
$\exists a, b, c, d \in \mathbb{B}. \ (a \lor b \lor \neg c) \land (\neg b \lor d)$

find bool $a, b, c, d$ such that $(a \lor b \lor \neg c) \land (\neg b \lor d)$
Satisfiability?

\[ \exists a, b, c, d \in \mathbb{B}. \ (a \lor b \lor \neg c) \land (\neg b \lor d) \]

find bool a, b, c, d such that (a || b || !c) && (!b || d)

- are there packages that satisfy all dependencies?
- is there an input that leads to a segfault?
- are there values where an LLVM optimization is incorrect?
- is there an unexpected way to access an S3 bucket?
bool is not enough

- package versions (→ 0.8.15)
- program variables (→ 42, "foobar", 0.12345)
- pattern matching (→ "arn:aws:ec2:*:*:instance/*")
bool is not enough

- package versions \(\rightarrow 0.8.15\)
- program variables \(\rightarrow 42, \"foobar\", 0.12345\)
- pattern matching \(\rightarrow \text{"arn:aws:ec2:*:*:instance/*"}\)

boolean **expression** instead of boolean **variable**

- \(0.8.15 \leq x \leq 1.0.0\)
- \(x \times x > y + 1\)
- \(0.123 + x = 0.345 \mid y\)
- \(\text{concat}(x, \"bar\") = \"foobar\"\)
- \(\text{matches}(r\"foo.*bar\", \"foobaz\")\)

\[
\exists a \in \mathbb{BV}_{64}, b \in \mathbb{FP}_{64}. \quad \text{tofp}(\text{bvxor}(a, \text{toubv}(b))) > b
\]
SMT solving in a nutshell

\[ \varphi \]

SAT solver

SAT or UNSAT

SAT solver

SAT + witness

or

UNSAT + reason

Theory solvers

theory constraints

Boolean model
SMT solving in a nutshell

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]
SMT solving in a nutshell

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]

\{ x > 0, x^2 > 0 \}

SAT solver

SAT or UNSAT

SAT + witness

or

UNSAT + reason
SMT solving in a nutshell

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]

\{x > 0, x^2 > 0\} \rightarrow \text{SAT} + x \mapsto 1

\text{SAT solver} \rightarrow \text{SAT or UNSAT}
SMT solving in a nutshell

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]

\{ x > 0, x^2 > 0, x^3 < 0 \}

SAT solver \rightarrow SAT or UNSAT

SAT + x \mapsto 1

Theory solvers
SMT solving in a nutshell

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]

\[ \{x > 0, x^2 > 0, x^3 < 0\} \quad \text{UNSAT} + \{x > 0, x^3 < 0\} \]

SAT solver \rightarrow SAT or UNSAT

Theory solvers
SMT solving in a nutshell

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0) \]

\{ x > 0, x^2 > 0, x^3 < 0 \} \rightarrow \text{UNSAT} + \{ x > 0, x^3 < 0 \}

SAT solver \rightarrow \text{SAT or UNSAT}

Theory solvers
SMT solving in a nutshell

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0) \]

SAT solver

\{ x > 0, \neg x^3 < 0, x = 3 \} \quad \text{UNSAT} + \{ x > 0, x^3 < 0 \}

Theory solvers

SAT or UNSAT
\( x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0) \)
SMT solving in a nutshell

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0) \]

\{ x > 0, \neg x^3 < 0, x = 3, x^2 > 0 \} \quad \text{SAT} + x \mapsto 3

SAT solver

SAT or UNSAT

Theory solvers
SMT solving in a nutshell

$x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0)$

SAT solver

SAT, $x \mapsto 3$

$\{x > 0, \neg x^3 < 0, x = 3, x^2 > 0\}$

SAT + $x \mapsto 3$

Theory solvers
rough overview

- SAT: NP complete $O(2^n)$
- UF: SAT + congruence closure
- AX: via UF, limited overhead
- BV: via SAT, sometimes quadratic formula growth $O(2^{n^2})$
- FP: via BV, formula growth, all bits significant $+\varepsilon$
- LRA: SAT + simplex $+O(2^n)$
- LIA: SAT + simplex + integrality $+O(2^{2^n})$
- NRA: SAT + computer algebra $+O(2^{2^n})$
- NIA: undecidable
- S: almost immediately undecidable
- ...

... but: (surprisingly?) good performance in practice!
Hard?

rough overview

- SAT: NP complete \( O(2^n) \)
- UF: SAT + congruence closure
- AX: via UF, limited overhead
- BV: via SAT, sometimes quadratic formula growth \( O(2^{n^2}) \)
- FP: via BV, formula growth, all bits significant +\( \varepsilon \)
- LRA: SAT + simplex +\( O(2^n) \)
- LIA: SAT + simplex + integrality
- NRA: SAT + computer algebra +\( O(2^{2^n}) \)
- NIA: undecidable
- S: almost immediately undecidable
- ...

but: (surprisingly?) good performance in practice!
Formal verification?

Formal guarantees

- no statistical guarantees
- no “probably correct”
- no “we haven’t found anything”
- no “that’s close to a solution”
- no “it works, except in these cases”

Solver says

- sat: the model satisfies the formula
- unsat: there is no model
- (unknown for undecidable logics and incomplete theory solvers)
- otherwise file a bug!
Beyond satisfiability

- variable assignments
- unsat cores
- quantifiers
- optimization
- interpolants
- formal proofs
- synthesis
- ...
SMT ecosystem

solvers usually open-source

- cvc5 (Stanford, Iowa)  
  [GitHub](https://github.com/cvc5/cvc5)
- yices (SRI)  
  [GitHub](https://github.com/SRI-CSL/yices2)
- z3 (Microsoft Research)  
  [GitHub](https://github.com/Z3Prover/z3)
- ... bitwuzla, colibri, dreal, iprover, ismt, mathsat, opensmt, ostrich, q3b, rasat, smtinterpol, smtrat, stp, vampire, yaga ...

SMT-COMP: yearly competition  
[GitHub](https://smt-comp.github.io)

SMT-LIB:  
[GitHub](https://smtlib.cs.uiowa.edu)

- benchmarks: >200k inputs from >80 logics
- input language: SMT-LIB 2.6, soon SMT-LIB 3.0
- tooling: syntax highlighting, parser, debugger, ...
int puts(const char *s) { ... }

int main(int argc, char **argv) {
    puts(argv[2]);
    return 0;
}

cbtc file.c --bounds-check --pointer-check ...

[main.pointer_dereference.6] line 3 dereference failure: pointer outside object bounds in argv[(signed long int)2]
C₀ production:

<table>
<thead>
<tr>
<th>ID</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Retrieve base with cap from shelf at CS</td>
</tr>
<tr>
<td>2</td>
<td>Prepare CS to retrieve cap</td>
</tr>
<tr>
<td>3</td>
<td>Feed base into CS</td>
</tr>
<tr>
<td>4</td>
<td>Prepare CS to mount cap</td>
</tr>
<tr>
<td>5</td>
<td>Feed black base to CS</td>
</tr>
<tr>
<td>6</td>
<td>Retrieve base from BS</td>
</tr>
<tr>
<td>7</td>
<td>Prepare BS to provide black base</td>
</tr>
<tr>
<td>8</td>
<td>Discard cap-less base</td>
</tr>
<tr>
<td>9</td>
<td>Retrieve black base with cap from CS</td>
</tr>
<tr>
<td>10</td>
<td>Prepare DS for slide specified in order</td>
</tr>
<tr>
<td>11</td>
<td>Deliver to DS</td>
</tr>
</tbody>
</table>

doi.org/10.1007/s10796-018-9858-3
float y = +0.0 - (-x);
float y_ = x; // equivalent?

// x = -0.0: y == +0.0, y_ == -0.0
// x = -0.0: 1/y == +inf, 1/y_ == -inf

**Alive + LifeJacket:**

- user implements LLVM optimization pass
- Alive encodes optimization into SMT formula
- z3 solves SMT formula
- 43 passes verified
- 8 bugs identified
Case study: Zephyrus 2 // automatic cloud deployments

Depoloyable Components

- wp_frontend
  - RAM: 7000
- wp_backend
  - RAM: 2000
- HTTP_LoadBalancer
  - mysql
- MySQL
  - RAM: 1000
  - $\geq 3$

Location (e.g., VMs, PCs, …)

- c3 large 1
  - RAM: 3750
  - Cost: 105
- c3 large 2
  - RAM: 3750
  - Cost: 105
- c3 large 3
  - RAM: 3750
  - Cost: 105
- c3 xlarge 1
  - RAM: 7000
  - Cost: 210

User Constraints

- WordPress
- MySQL
- HTTP_LoadBalancer
  - $\geq 3$

doi.org/10.1007/978-3-319-47677-3_15
Case study: Amazon // automatic policy checks

Runtime policy check with Zelkova

Event-driven lambda

1. Amazon S3 notification trigger
2. Identify resource policies
3. Look up baseline for the account
4. Invoke Zelkova to compare policies from snapshot vs. baseline
5. Send alert if more permissive
6. Auto-remediate to baseline policy

amazon.science/blog/a-billion-smt-queries-a-day
Case study: Certora // verification of smart contracts
Thanks!

Any questions?

gereon.kremer+froscon@gmail.com  gereon-kremer.de