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.

What?

openSUSE

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 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.

What?

openSUSE

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 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.

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.

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.

`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

Andres 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.

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.

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.

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

Andres 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.

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.

KLEE Symbolic Execution Engine

KLEE is a dynamic symbolic execution engine built on top of the LLVM compiler infrastructure, and available under the UIUC open source license. For more information on what KLEE is and what it can do, see the OSDI 2008 paper.

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.

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

Andres 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.

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.

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.

AUTOMATED REASONING

KLEE Symbolic Execution Engine

KLEE is a dynamic symbolic execution engine built on top of the LLVM compiler infrastructure, and available under the UIUC open source license. For more information on what KLEE is and what it can do, see the OSDI 2008 paper.

Satisfiability?

$$\exists a, b, c, d \in \mathbb{B}. \quad (a \vee b \vee \neg c) \wedge (\neg b \vee d)$$

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

Satisfiability?

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

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?

Modulo Theories?

`bool` is not enough

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

Modulo Theories?

`bool` is not enough

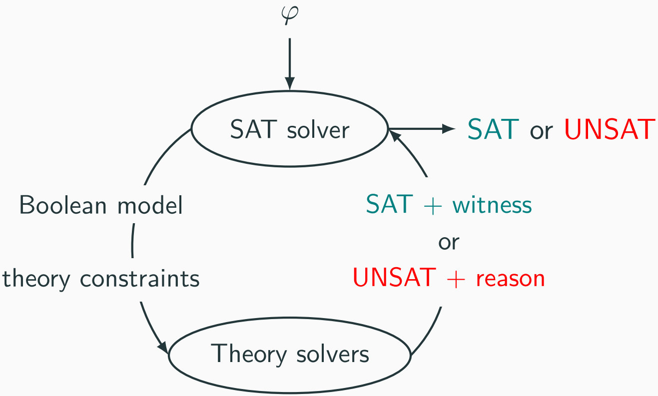
- package versions ($\rightarrow$ `0.8.15`)
- program variables ($\rightarrow$ `42`, `"foobar"`, `0.12345`)
- pattern matching ($\rightarrow$ `"arn:aws:ec2:*:*:instance/*"`)

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

- `0.8.15 <= x <= 1.0.0`
- `x * x > y + 1`
- `0.123 + x == 0.345 | y`
- `concat(x, "bar") == "foobar"`
- `matches(r"foo.*bar", "foobaz")`

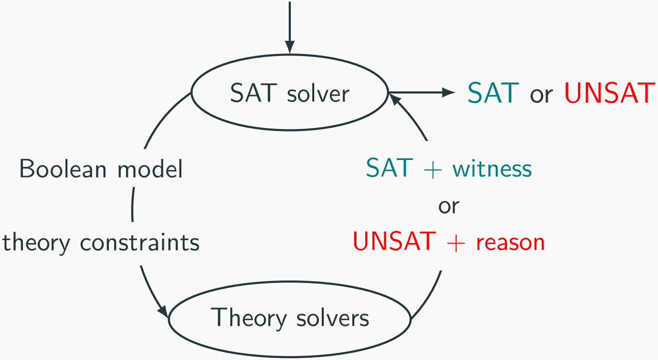
$$\exists a \in \mathbb{BV}_{64}, b \in \mathbb{FP}_{64}. \quad tofp(bvxor(a, toubv(b))) > b$$

SMT solving in a nutshell

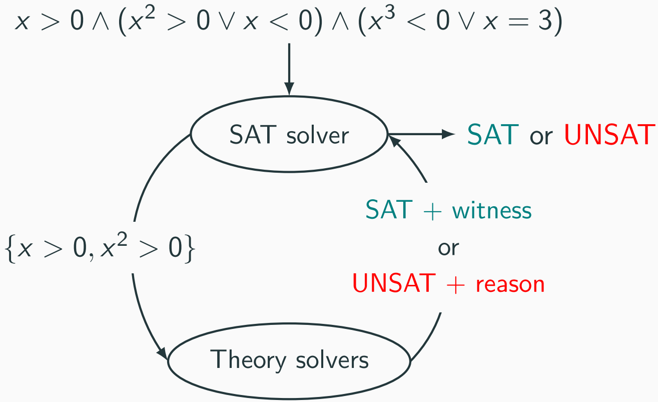


SMT solving in a nutshell

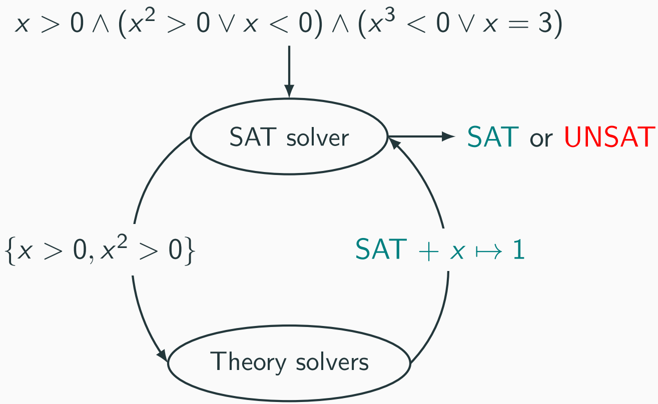
$$x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3)$$



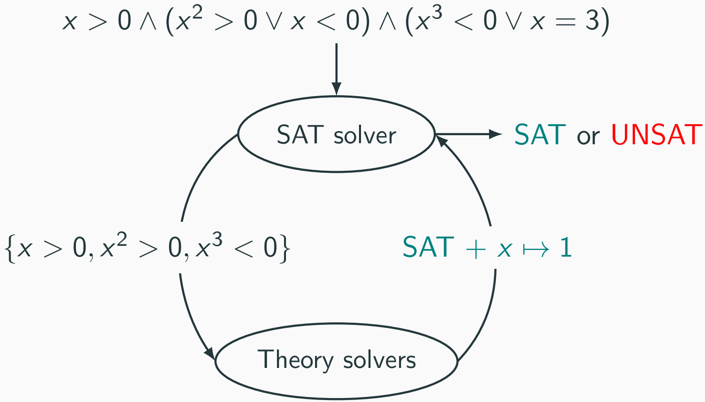
SMT solving in a nutshell



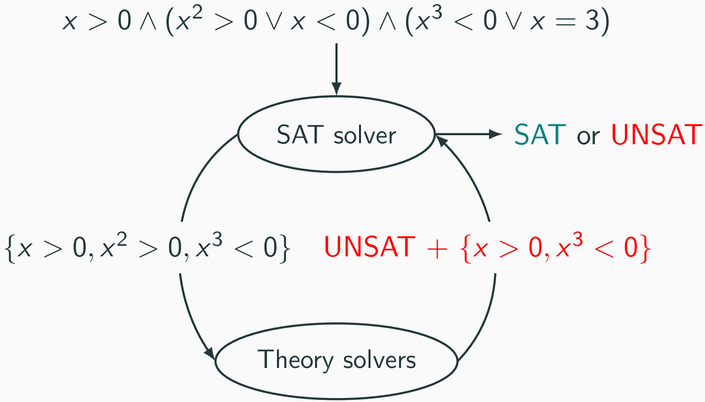
SMT solving in a nutshell



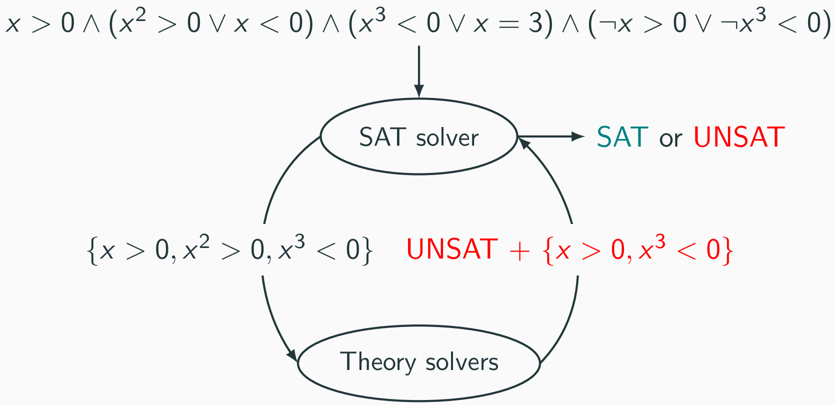
SMT solving in a nutshell



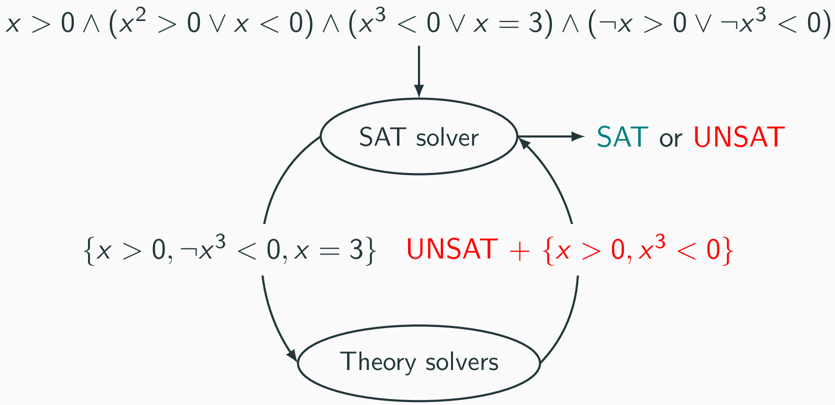
SMT solving in a nutshell



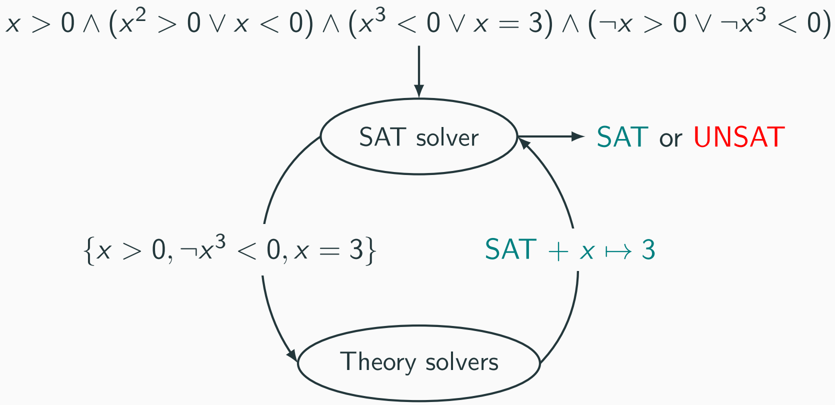
SMT solving in a nutshell



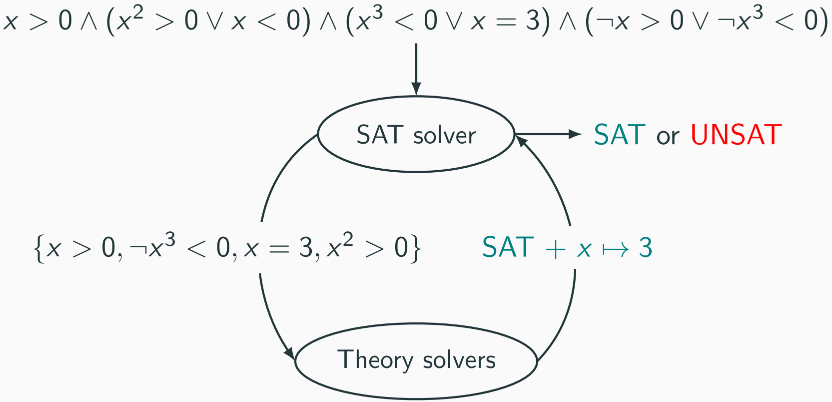
SMT solving in a nutshell



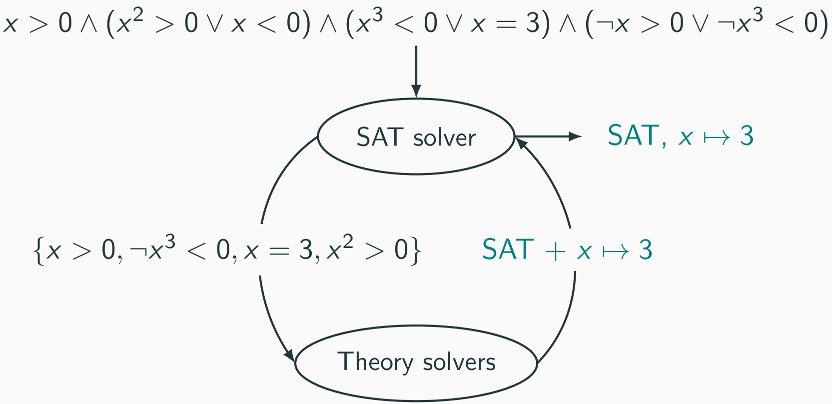
SMT solving in a nutshell



SMT solving in a nutshell



SMT solving in a nutshell



Hard?

rough overview | very simplified, borderline incorrect

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

Hard?

rough overview — very simplified, borderline incorrect

- SAT: NP complete — $\mathcal{O}(2^n)$
- UF: SAT + congruence closure
- AX: via UF, limited overhead
- BV: via SAT, sometimes quadratic formula growth — $\mathcal{O}(2^{n^2})$
- FP: via BV, formula growth, all bits significant — $+\varepsilon$
- LRA: SAT + simplex — $+\mathcal{O}(2^n)$
- LIA: SAT + simplex + integrality
- NRA: SAT + computer algebra — $+\mathcal{O}(2^{2^n})$
- 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 `x = ?`
- unsat cores why UNSAT?
- quantifiers $\forall x \exists y$
- optimization minimize $x * y$
- interpolants $\varphi_1 \Rightarrow \psi^? \Rightarrow \varphi_2$
- formal proofs verify UNSAT
- synthesis $\varphi(expr^?)$
- ...

SMT ecosystem

solvers — usually open-source

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

SMT-COMP: yearly competition — smt-comp.github.io

SMT-LIB: — smtlib.cs.uiowa.edu

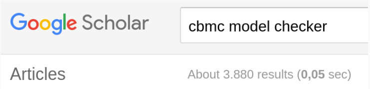
- benchmarks: >200k inputs from >80 logics
- input language: SMT-LIB 2.6, soon SMT-LIB 3.0
- tooling: syntax highlighting, parser, debugger, ...

Case study: CBMC // verification of C and C++

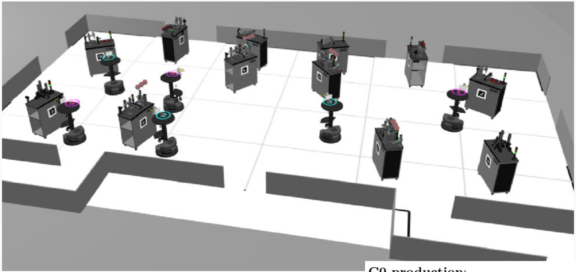
```c
int puts(const char *s) { ... }
int main(int argc, char **argv) {
  puts(argv[2]);
  return 0;
}
```

```
cbmc file.c --bounds-check --pointer-check ...
```

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

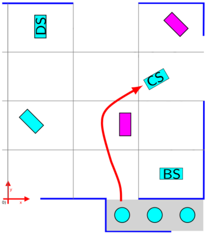


Case study: RoboCup Logistics // multi-robot planning



C0 production:

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



doi.org/10.1007/s10796-018-9858-3

Case study: LifeJacket // LLVM optimizations

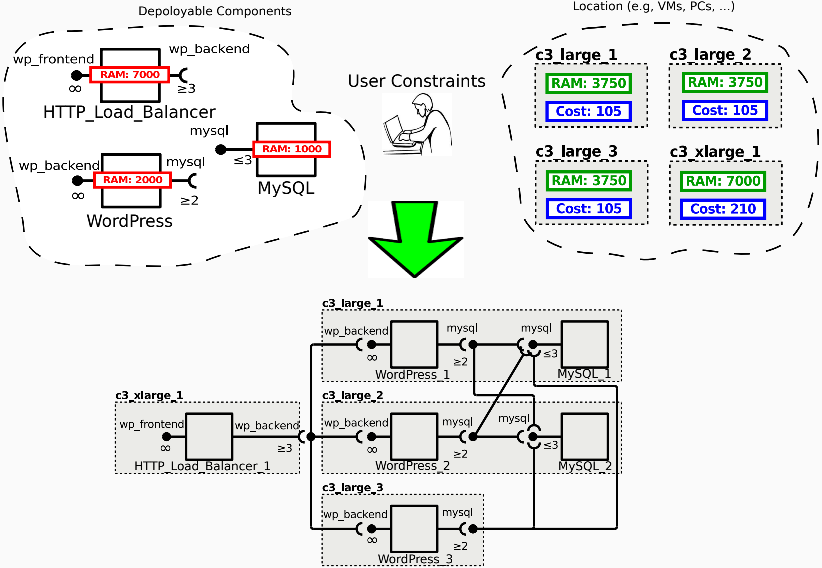
```
float y = +0.0 - (-x);
float y_ = x; // equivalent?

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

Alive + LifeJacket:

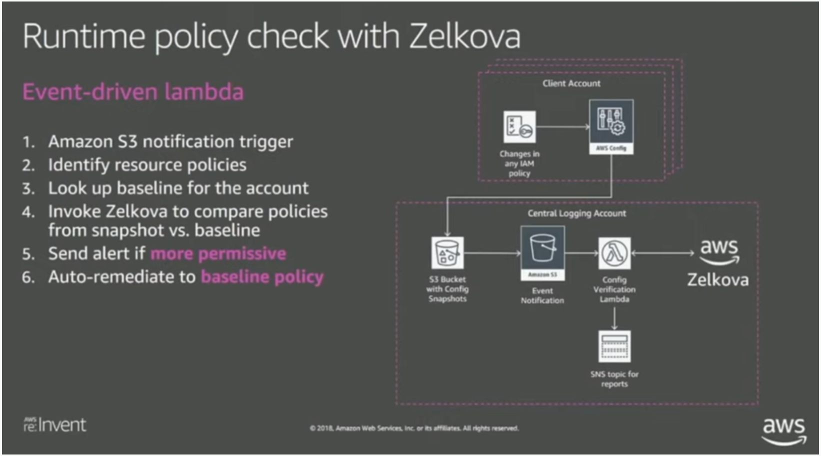
- user implements LLVM optimization pass
- Alive encodes optimization into SMT formula
- z3 solves SMT formula
- 43 passes verified
- 8 bugs identified

Case study: Zephyrus 2 // automatic cloud deployments



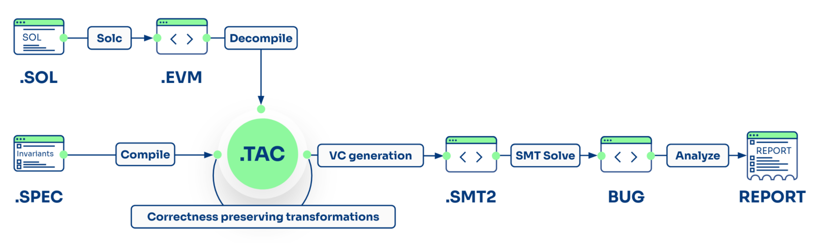
doi.org/10.1007/978-3-319-47677-3_15

Case study: Amazon // automatic policy checks



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