The How and Why of Higher-Order SMT for Prospective Users

Sophie Tourret

Journées Nationales du GDR GPL & AFADL
June 2024

SMT in Formal Methods

Z3, Alt-Ergo, cvc5, ...

SMT in Formal Methods

Z3, Alt-Ergo, cvc5, ...

SMT is Well-known as a backend for many techniques, including:

SMT in Formal Methods

Z3, Alt-Ergo, cvc5, ...

SMT is Well-known as a backend for many techniques, including:

- program verification (Boogie, F*, Viper, Why3, Frama-C, Atelier-B...)

SMT in Formal Methods

Z3, Alt-Ergo, cvc5, ...

SMT is Well-known as a backend for many techniques, including:

- program verification (Boogie, F*, Viper, Why3, Frama-C, Atelier-B...)
- symbolic execution (KLEE, S2E, Triton)

SMT in Formal Methods

Z3, Alt-Ergo, cvc5, ...

SMT is Well-known as a backend for many techniques, including:

- program verification (Boogie, F*, Viper, Why3, Frama-C, Atelier-B...)
- symbolic execution (KLEE, S2E, Triton)
- interactive proof assistants (Isabelle/HOL, Coq, HOL)

Standard SMT Solving

The Bases (1/2)

SMT stands for Satisfiability Modulo Theories

The Bases (1/2)

SMT stands for Satisfiability Modulo Theories

An SMT solver determins the truth value of a formula.

A formula is . . .

The Bases (1/2)

SMT stands for Satisfiability Modulo Theories

An SMT solver determins the truth value of a formula.

A formula is . . .

valid when always true,

The Bases (1/2)

SMT stands for Satisfiability Modulo Theories

An SMT solver determins the truth value of a formula.

A formula is ...

valid when always true,

satisfiable when true at least once,

The Bases (1/2)

SMT stands for Satisfiability Modulo Theories

An SMT solver determins the truth value of a formula.

A formula is . . .

valid when always true,

satisfiable when true at least once,

unsatisfiable when never true.

The Bases (2/2)

SMT solvers usually operate in first-order logic

$+$ interpreted symbols in given theories

The Bases (2/2)

SMT solvers usually operate in first-order logic

- formula ϕ, ψ: built from $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \ldots$ and quantifiers
- quantifiers $\forall$,$\exists$: $\forall x.\phi$, $\exists y.\psi$
- bound variables: $\forall x, y. P(f(x), y) \vee Q(y)$

$+$ interpreted symbols in given theories

The Bases (2/2)

SMT solvers usually operate in first-order logic

- formula ϕ, ψ: built from $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \ldots$ and quantifiers
- quantifiers $\forall$,$\exists$: $\forall x.\phi$, $\exists y.\psi$
- bound variables: $\forall x, y. P(f(x), y) \vee Q(y)$

$+$ interpreted symbols in given theories

- $+$, $\times$, $\leq$, $=$, $\ldots$

The Bases (2/2)

SMT solvers usually operate in first-order logic

- formula ϕ, ψ: built from $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \ldots$ and quantifiers
- quantifiers $\forall$,$\exists$: $\forall x.\phi$, $\exists y.\psi$
- bound variables: $\forall x, y.P(f(x), y) \vee Q(y)$

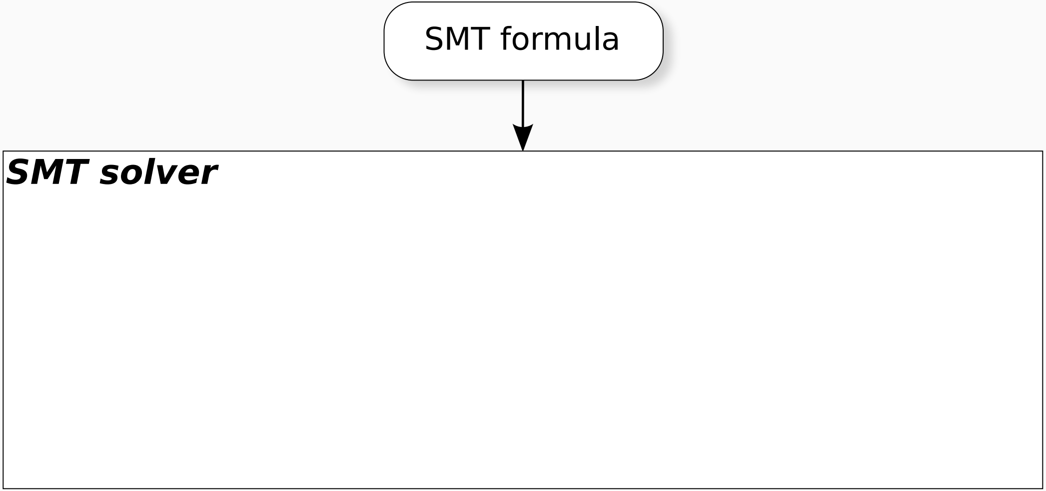
+ interpreted symbols in given theories

- $+$, $\times$, $\leq$, $=$, $\ldots$

Example

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

Inside an SMT solver



SMT Inputs

Returning to our example:

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

SMT Inputs

Returning to our example:

$$a \leq b \land b \leq a + c \land c = 0 \land \left[a \neq b \lor (q(a) \land \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \land b \leq a + c \land c = 0 \land \left[a \neq b \lor (q(a) \land \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \land b \leq a + c \land c = 0 \land \left[a \neq b \lor (q(a) \land \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \land b \leq a + c \land c = 0 \land \left[a \neq b \lor (q(a) \land \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \land b \leq a + c \land c = 0 \land \left[a \neq b \lor \left(q(a) \land \neg q(f(b) + c)\right)\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

Returning to our example:

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

SMT Inputs

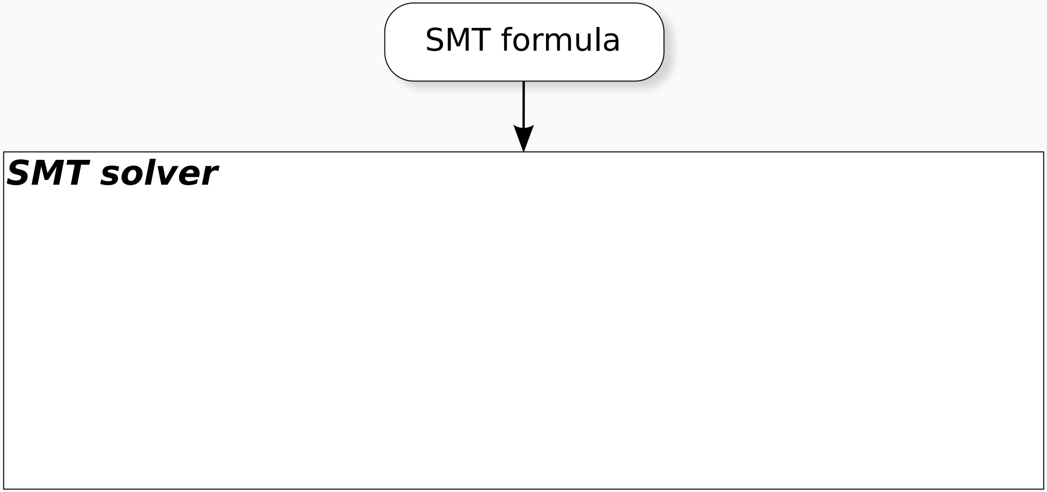
Returning to our example:

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

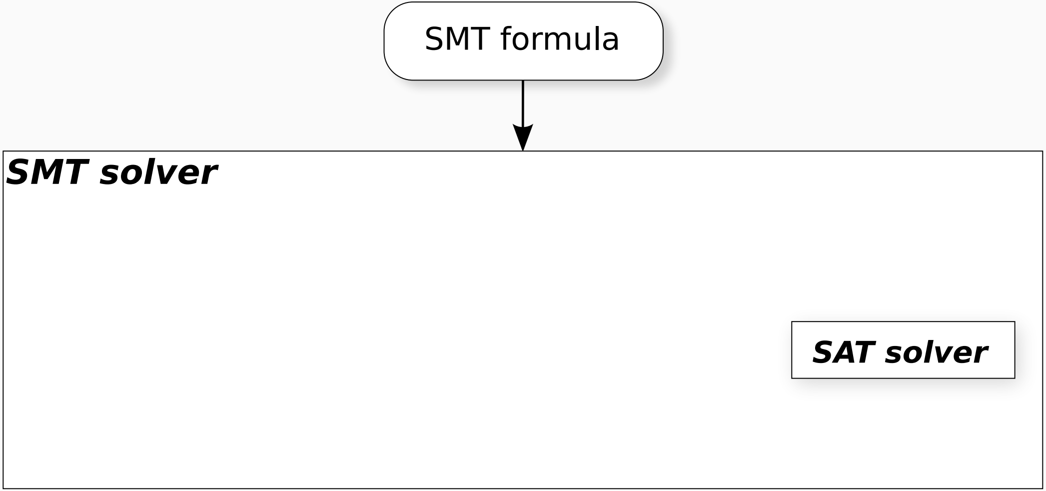
encoded in SMT-LIB 2.0 format:

```
(set-logic QF_UFLIA)
(set-info :source | Example formula in SMT-LIB 2.6 |)
(set-info :smt-lib-version 2.6)
(declare-fun f (Int) Int)
(declare-fun q (Int) Bool)
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)
(assert (and (<= a b) (<= b (+ a c)) (= c 0)
  (or (not (= a b))
    (and (q a) (not (q (+ (f b) c)))))))
(check-sat)
```

Inside an SMT solver



Inside an SMT solver



SAT Solving

Many solvers: CaDiCal, Kissat, SAT4J, MiniSAT, Glucose, Crypto-MiniSAT . . .

Many uses:

- for cryptography

SAT Solving

Many solvers: CaDiCal, Kissat, SAT4J, MiniSAT, Glucose, Crypto-MiniSAT ...

Many uses:

- for cryptography
- for teaching

SAT Solving

Many solvers: CaDiCal, Kissat, SAT4J, MiniSAT, Glucose, Crypto-MiniSAT ...

Many uses:

- for cryptography
- for teaching
- for parallel computation

SAT Solving

Many solvers: CaDiCal, Kissat, SAT4J, MiniSAT, Glucose, Crypto-MiniSAT . . .

Many uses:

- for cryptography
- for teaching
- for parallel computation
- for cloud computation

SAT Solving

Many solvers: CaDiCal, Kissat, SAT4J, MiniSAT, Glucose, Crypto-MiniSAT ...

Many uses:

- for cryptography
- for teaching
- for parallel computation
- for cloud computation
- for incremental computation

SAT Solving

Many solvers: CaDiCal, Kissat, SAT4J, MiniSAT, Glucose, Crypto-MiniSAT . . .

Many uses:

- for cryptography
- for teaching
- for parallel computation
- for cloud computation
- for incremental computation

SAT Solving

Many solvers: CaDiCal, Kissat, SAT4J, MiniSAT, Glucose, Crypto-MiniSAT . . .

Many uses:

- for cryptography
- for teaching
- for parallel computation
- for cloud computation
- for incremental computation

Interface standardization efforts:

- IPASIR, well-established

SAT Solving

Many solvers: CaDiCal, Kissat, SAT4J, MiniSAT, Glucose, Crypto-MiniSAT ...

Many uses:

- for cryptography
- for teaching
- for parallel computation
- for cloud computation
- for incremental computation

Interface standardization efforts:

- IPASIR, well-established
- IPASIR-UP, new, designed for SMT

SAT Solving

Many solvers: CaDiCal, Kissat, SAT4J, MiniSAT, Glucose, Crypto-MiniSAT . . .

Many uses:

- for cryptography
- for teaching
- for parallel computation
- for cloud computation
- for incremental computation

Interface standardization efforts:

- IPASIR, well-established
- IPASIR-UP, new, designed for SMT
- IPASIR-2, to come, independent from IPASIR-UP but synergies

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver.

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver. It must be abstracted, e.g.,

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver. It must be abstracted, e.g.,

$$P \quad \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \leq b \land b \leq a + c \land c = 0 \land \left[a \neq b \lor (q(a) \land \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver. It must be abstracted, e.g.,

$$P \quad \land \qquad Q \quad \land c = 0 \land \left[a \neq b \lor (q(a) \land \neg q(f(b) + c))\right]$$

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \leq b \land b \leq a + c \land c = 0 \land \left[a \neq b \lor (q(a) \land \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver. It must be abstracted, e.g.,

$$P \quad \land \qquad Q \quad \land \quad R \quad \land \left[a \neq b \lor (q(a) \land \neg q(f(b) + c))\right]$$

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver. It must be abstracted, e.g.,

$$P \quad \wedge \qquad Q \quad \wedge \quad R \quad \wedge \left[\quad \neg S \quad \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \le b \wedge b \le a + c \wedge c = 0 \wedge \left[a \ne b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver. It must be abstracted, e.g.,

$$P \quad \wedge \qquad Q \quad \wedge \quad R \quad \wedge \left[\neg S \quad \vee (\quad T \quad \wedge \neg q(f(b) + c))\right]$$

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \leq b \land b \leq a + c \land c = 0 \land \left[a \neq b \lor (q(a) \land \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver. It must be abstracted, e.g.,

$$P \quad \land \qquad Q \quad \land \quad R \quad \land \left[\; \neg S \quad \lor (\quad T \; \land \qquad \neg U \qquad)\right]$$

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver. It must be abstracted, e.g.,

$$P \quad \wedge \qquad Q \quad \wedge \quad R \quad \wedge \left[\ \neg S \quad \vee (\quad T \ \wedge \qquad \neg U \qquad)\right]$$

If the abstracted formula is UNSAT, so is the SMT formula.

SAT Solving for SMT

An SMT formula, e.g., our running example

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[a \neq b \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

cannot be handled by a SAT solver. It must be abstracted, e.g.,

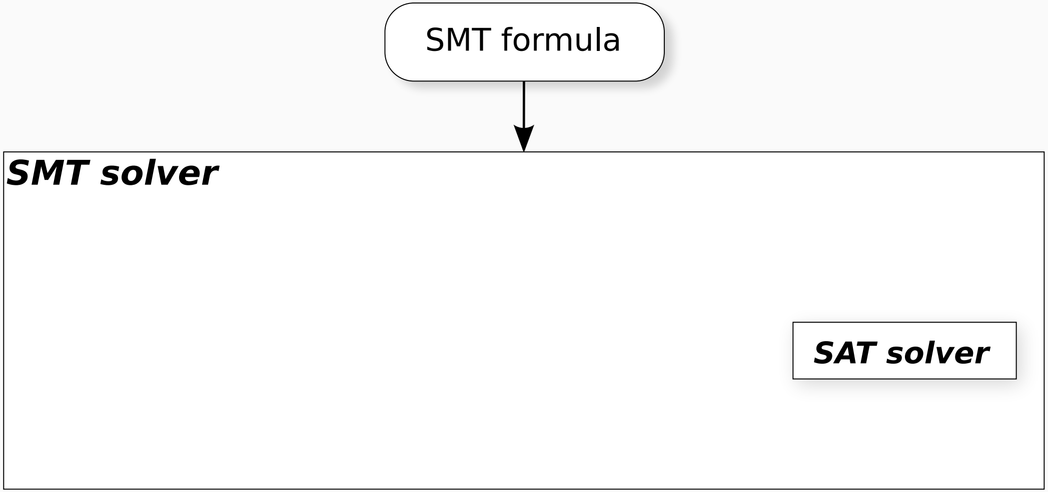
$$P \quad \wedge \qquad Q \quad \wedge \quad R \quad \wedge \left[\ \neg S \quad \vee (\quad T \ \wedge \qquad\qquad \neg U \qquad)\right]$$

If the abstracted formula is UNSAT, so is the SMT formula.

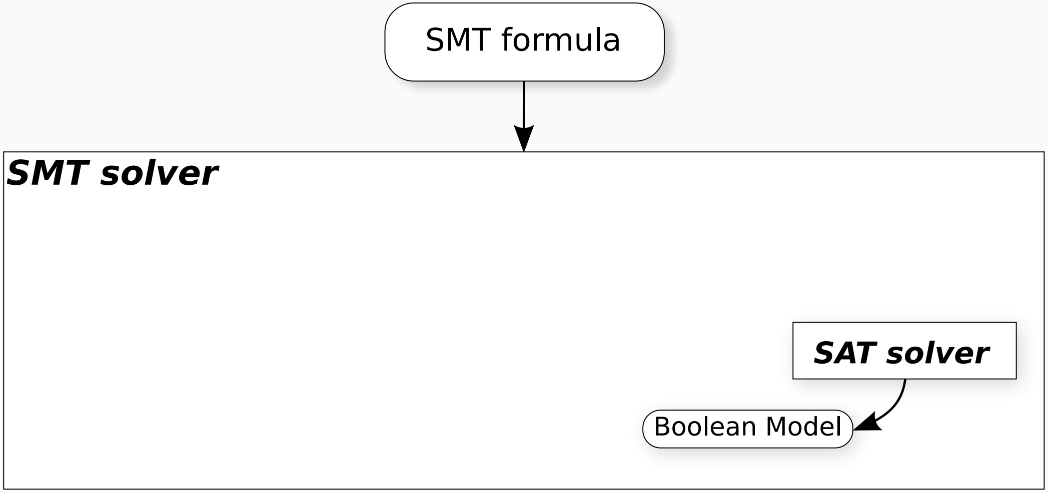
Otherwise the SAT solver provides a model to the SMT solver, e.g.,

$$P \wedge Q \wedge R \wedge \neg S$$

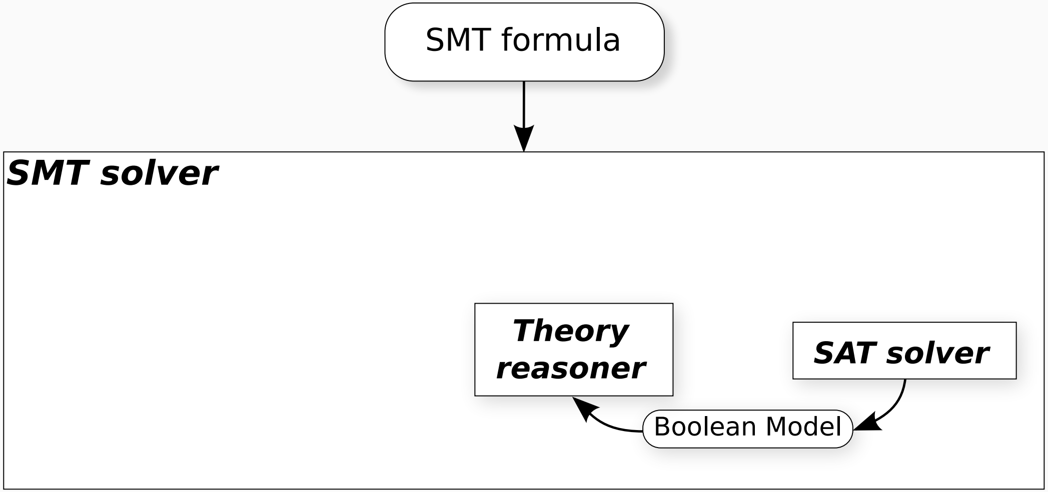
Inside an SMT solver



Inside an SMT solver



Inside an SMT solver



First-order Theories

The most useful theories for verification include:

Equality:

Equality with uninterpreted symbols (EUF) congruence closure $f(x) = y$, $g(a, b) = a$

First-order Theories

The most useful theories for verification include:

Equality:

Equality with uninterpreted symbols (EUF)	congruence closure	$f(x) = y$, $g(a, b) = a$

Math:

linear arithmetic (real, integers) (LIA, LRA)	mostly simplex	$x + 3y = 22$
non-linear arithmetic	CAD, Gröbner bases...	$3x^2 + 2x - 8 = 0$

First-order Theories

The most useful theories for verification include:

Equality:

Equality with uninterpreted symbols (EUF)	congruence closure	$f(x) = y$, $g(a, b) = a$

Math:

linear arithmetic (real, integers) (LIA, LRA)	mostly simplex	$x + 3y = 22$
non-linear arithmetic	CAD, Gröbner bases...	$3x^2 + 2x - 8 = 0$

Data structures:

arrays	uninterpreted symbols	read(a,i) = b
bitvectors	bit-blasting	concat bv_i $bv_j = bv_m$
strings	SAT + arithmetic	"a" · "bc" = "ab" · "c"

Theories for SMT

Theory solvers detect problematic assignments done by the SAT solver, e.g.,

Theories for SMT

Theory solvers detect problematic assignments done by the SAT solver, e.g., if the SAT solver found the model

$$P \wedge Q \wedge R \wedge \neg S$$

for our running example, it means

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge a \neq b.$$

Theories for SMT

Theory solvers detect problematic assignments done by the SAT solver, e.g., if the SAT solver found the model

$$P \wedge Q \wedge R \wedge \neg S$$

for our running example, it means

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge a \neq b.$$

Then an LIA solver finds that both $a = b$ and $a \neq b$ must hold and returns false.

Theories for SMT

Theory solvers detect problematic assignments done by the SAT solver, e.g., if the SAT solver found the model

$$P \wedge Q \wedge R \wedge \neg S$$

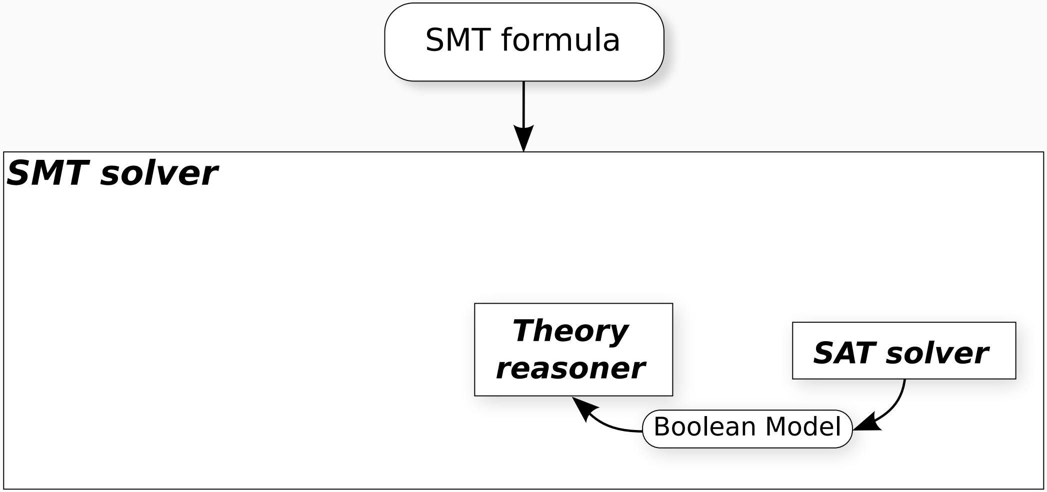
for our running example, it means

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge a \neq b.$$

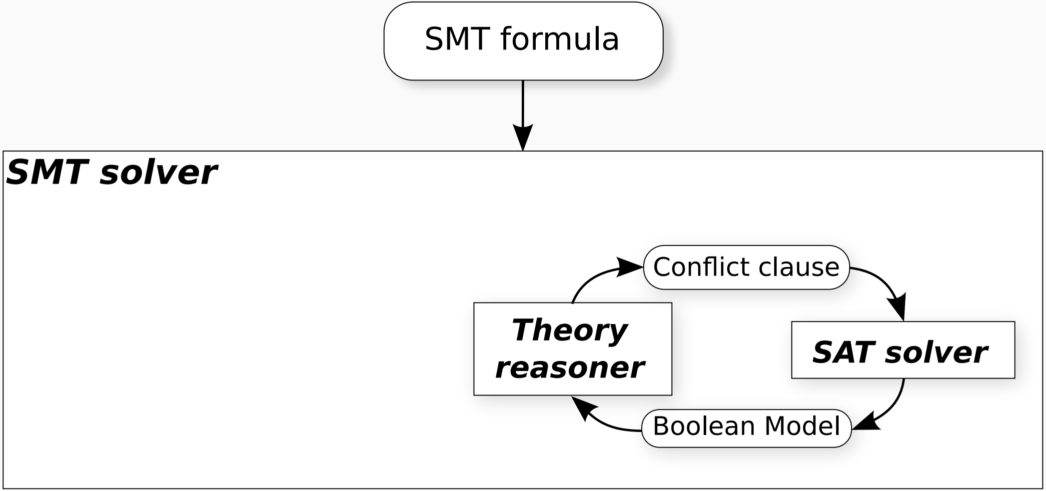
Then an LIA solver finds that both $a = b$ and $a \neq b$ must hold and returns false.

The formula $\neg P \vee \neg Q \vee \neg R \vee S$ is added to the abstracted formula before calling the SAT solver once more.

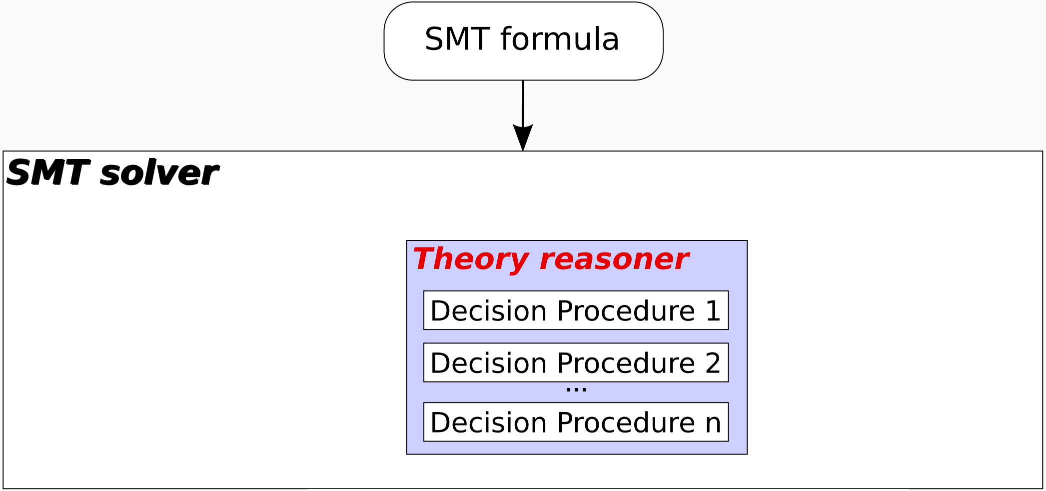
Inside an SMT solver



Inside an SMT solver



Inside an SMT solver



Combining Theories

If our example,

$$P \wedge Q \wedge R \wedge \neg S$$

means in fact

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge f(a) \neq f(b).$$

Combining Theories

If our example,

$$P \land Q \land R \land \neg S$$

means in fact

$$a \leq b \land b \leq a + c \land c = 0 \land f(a) \neq f(b).$$

Combining Theories

If our example,

$$P \wedge Q \wedge R \wedge \neg S$$

means in fact

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge f(a) \neq f(b).$$

Both LIA and EUF are needed. How to combine them?

Combining Theories

If our example,

$$P \wedge Q \wedge R \wedge \neg S$$

means in fact

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge f(a) \neq f(b).$$

Both LIA and EUF are needed. How to combine them?

By exchanging equations and disequations, e.g.,

- LIA: $a \leq b$, $b \leq a + c$, $c = 0$
- EUF: $f(a) \neq f(b)$

Combining Theories

If our example,

$$P \wedge Q \wedge R \wedge \neg S$$

means in fact

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge f(a) \neq f(b).$$

Both LIA and EUF are needed. How to combine them?

By exchanging equations and disequations, e.g.,

- LIA: $a \leq b$, $b \leq a + c$, $c = 0 \quad \Longrightarrow b \leq a$
- EUF: $f(a) \neq f(b)$

Combining Theories

If our example,

$$P \land Q \land R \land \neg S$$

means in fact

$$a \leq b \land b \leq a + c \land c = 0 \land f(a) \neq f(b).$$

Both LIA and EUF are needed. How to combine them?

By exchanging equations and disequations, e.g.,

- LIA: $a \leq b$, $b \leq a + c$, $c = 0 \quad \Longrightarrow b \leq a \quad \Longrightarrow a = b$
- EUF: $f(a) \neq f(b)$

Combining Theories

If our example,

$$P \wedge Q \wedge R \wedge \neg S$$

means in fact

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge f(a) \neq f(b).$$

Both LIA and EUF are needed. How to combine them?

By exchanging equations and disequations, e.g.,

- LIA: $a \leq b,\ b \leq a + c,\ c = 0 \quad \Longrightarrow b \leq a \quad \Longrightarrow a = b$
- EUF: $f(a) \neq f(b)$

Combining Theories

If our example,

$$P \land Q \land R \land \neg S$$

means in fact

$$a \leq b \land b \leq a + c \land c = 0 \land f(a) \neq f(b).$$

Both LIA and EUF are needed. How to combine them?

By exchanging equations and disequations, e.g.,

- LIA: $a \leq b,\ b \leq a + c,\ c = 0 \quad \Longrightarrow b \leq a \quad \Longrightarrow a = b$
- EUF: $f(a) \neq f(b),\ a = b$

Combining Theories

If our example,

$$P \land Q \land R \land \neg S$$

means in fact

$$a \leq b \land b \leq a + c \land c = 0 \land f(a) \neq f(b).$$

Both LIA and EUF are needed. How to combine them?

By exchanging equations and disequations, e.g.,

- LIA: $a \leq b,\ b \leq a + c,\ c = 0 \quad \Longrightarrow b \leq a \quad \Longrightarrow a = b$
- EUF: $f(a) \neq f(b),\ a = b \quad \Longrightarrow a \neq b$

Combining Theories

If our example,

$$P \wedge Q \wedge R \wedge \neg S$$

means in fact

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge f(a) \neq f(b).$$

Both LIA and EUF are needed. How to combine them?

By exchanging equations and disequations, e.g.,

- LIA: $a \leq b$, $b \leq a + c$, $c = 0 \quad \Longrightarrow b \leq a \quad \Longrightarrow a = b$
- EUF: $f(a) \neq f(b)$, $a = b \quad \Longrightarrow a \neq b \quad \Longrightarrow$ contradiction!

Combining Theories

If our example,

$$P \land Q \land R \land \neg S$$

means in fact

$$a \leq b \land b \leq a + c \land c = 0 \land f(a) \neq f(b).$$

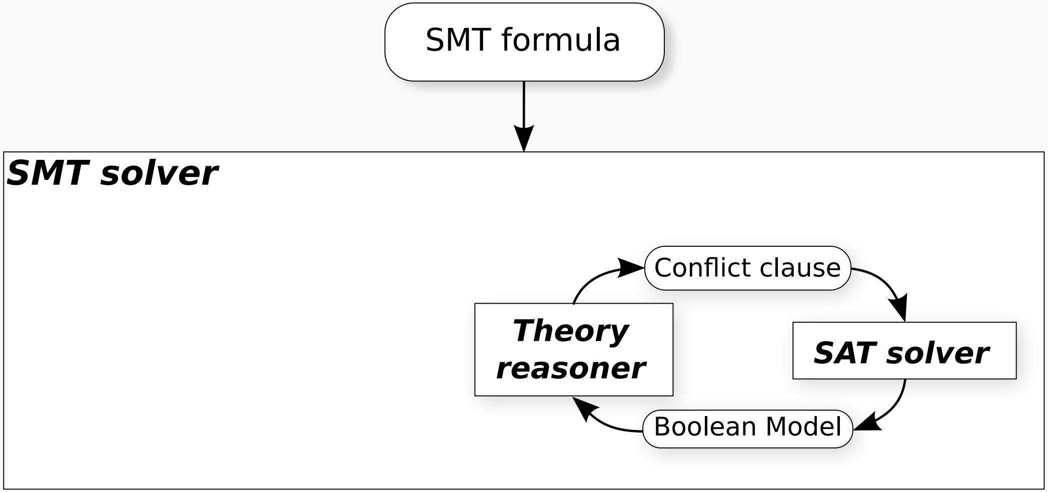
Both LIA and EUF are needed. How to combine them?

By exchanging equations and disequations, e.g.,

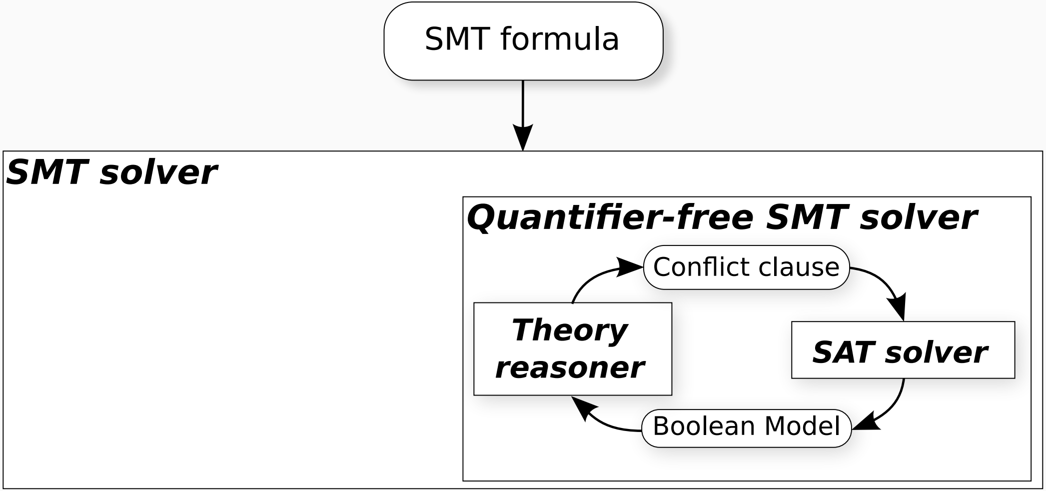
- LIA: $a \leq b,\ b \leq a + c,\ c = 0 \quad \Longrightarrow b \leq a \quad \Longrightarrow a = b$
- EUF: $f(a) \neq f(b),\ a = b \quad \Longrightarrow a \neq b \quad \Longrightarrow$ contradiction!

Various techniques: Nelson-Open, Shostak, Gentleness, Politeness, ...

Inside an SMT solver



Inside an SMT solver



Quantified Formulas in SMT (1/3)

Let us add to our improved running example,

$$a \leq b \land b \leq a + c \land c = 0 \land \left[f(a) \neq f(b) \lor (q(a) \land \neg q(f(b) + c)) \right]$$

the quantified formula

$$\forall x, y.\, (q(y) \implies q(g(y) + x))$$

Quantified Formulas in SMT (1/3)

Let us add to our improved running example,

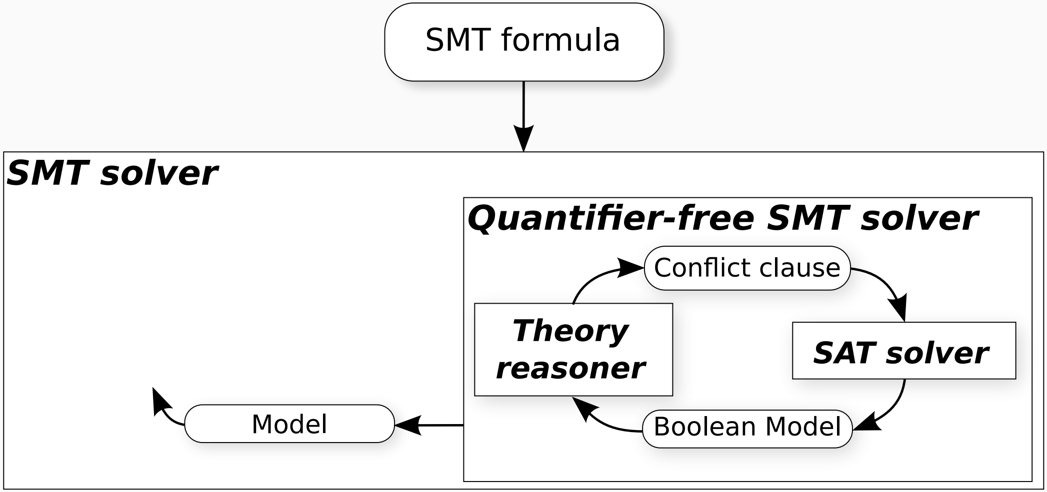
$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

the quantified formula

$$\forall x, y. \left(q(y) \implies q(g(y) + x)\right)$$

First the ground SMT solver will be queried for a model

Inside an SMT solver



Quantified Formulas in SMT (2/3)

If our running example,

$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(f(b) + c))\right]$$

also includes the formula

$$\forall x, y. \, (q(y) \implies q(g(y) + x))$$

First the ground SMT solver will be queried for a model

Quantified Formulas in SMT (2/3)

If our running example,

$$a \leq b \land b \leq a + c \land c = 0 \land \left[f(a) \neq f(b) \lor (q(a) \land \neg q(f(b) + c))\right]$$

also includes the formula

$$\forall x, y.\, (q(y) \implies q(g(y) + x))$$

First the ground SMT solver will be queried for a model, here

$$a \leq b, b \leq a + c, c = 0, q(a), \neg q(f(b) + c)$$

Quantified Formulas in SMT (2/3)

If our running example,

$$a \leq b \land b \leq a + c \land c = 0 \land \left[f(a) \neq f(b) \lor (q(a) \land \neg q(f(b) + c))\right]$$

also includes the formula

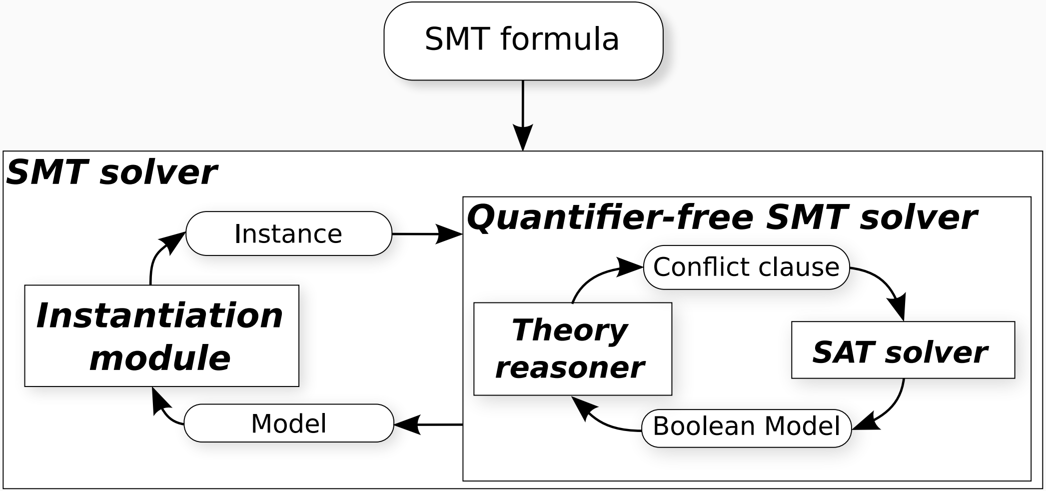
$$\forall x, y.\, (q(y) \implies q(g(y) + x))$$

First the ground SMT solver will be queried for a model, here

$$a \leq b, b \leq a + c, c = 0, q(a), \neg q(f(b) + c)$$

Then instances of the non-ground formulas will be produced based on this model and fed to the ground SMT solver.

Inside an SMT solver



Quantified Formulas in SMT (3/3)

for $$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(f(b) + c))\right]$$
$$\forall x, y.\, (q(y) \implies q(f(y) + x))$$

given the model $a \leq b, b \leq a + c, c = 0, q(a), \neg q(g(b) + c)$

Quantified Formulas in SMT (3/3)

for
$$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(f(b) + c))\right]$$
$$\forall x, y.\, (q(y) \implies q(f(y) + x))$$

given the model $a \leq b, b \leq a + c, c = 0, q(a), \neg q(g(b) + c)$

The instance where $y \mapsto a$ and $x \mapsto f(b) - g(a)$, i.e.,

$$q(a) \implies q(g(a) + f(b) - g(a))$$

Quantified Formulas in SMT (3/3)

for $$a \leq b \wedge b \leq a + c \wedge c = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(f(b) + c))\right]$$
$$\forall x, y.\, (q(y) \Longrightarrow q(f(y) + x))$$

given the model $a \leq b, b \leq a + c, c = 0, q(a), \neg q(g(b) + c)$

The instance where $y \mapsto a$ and $x \mapsto f(b) - g(a)$, i.e.,

$$q(a) \Longrightarrow q(g(a) + f(b) - g(a))$$

leads to a contradiction at the ground level!

Instantiation Techniques

There is no panacea!

Instantiation Techniques

There is no panacea!

Instantiation techniques:

- trigger-based

Instantiation Techniques

There is no panacea!

Instantiation techniques:

- trigger-based heuristic, to find unsat

Instantiation Techniques

There is no panacea!

Instantiation techniques:

- trigger-based — heuristic, to find unsat
- conflict-based

Instantiation Techniques

There is no panacea!

Instantiation techniques:

- trigger-based heuristic, to find unsat
- conflict-based also heuristic, to find unsat

Instantiation Techniques

There is no panacea!

Instantiation techniques:

- trigger-based — heuristic, to find unsat
- conflict-based — also heuristic, to find unsat, very efficient when it works

Instantiation Techniques

There is no panacea!

Instantiation techniques:

- trigger-based — heuristic, to find unsat
- conflict-based — also heuristic, to find unsat, very efficient when it works
- model-based

Instantiation Techniques

There is no panacea!

Instantiation techniques:

- trigger-based — heuristic, to find unsat
- conflict-based — also heuristic, to find unsat, very efficient when it works
- model-based — complete for decidable fragments, to find sat

Instantiation Techniques

There is no panacea!

Instantiation techniques:

- trigger-based — heuristic, to find unsat
- conflict-based — also heuristic, to find unsat, very efficient when it works
- model-based — complete for decidable fragments, to find sat
- enumerative

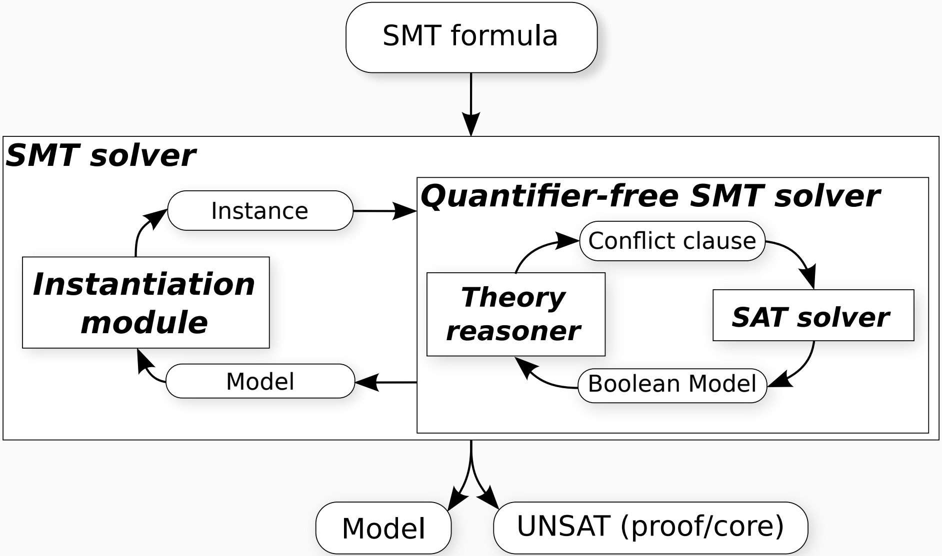
Instantiation Techniques

There is no panacea!

Instantiation techniques:

- trigger-based — heuristic, to find unsat
- conflict-based — also heuristic, to find unsat, very efficient when it works
- model-based — complete for decidable fragments, to find sat
- enumerative — complete for finitely populated types

Inside an SMT solver



SMT Solving in Higher-Order Logic

Higher-Order Logic (HOL)

- functional variables $y\ a = g\ a\ b$

Higher-Order Logic (HOL)

- functional variables $y\, a = g\, a\, b$
- partially applied functions $g\, a = f$

Higher-Order Logic (HOL)

- functional variables $y\, a = g\, a\, b$
- partially applied functions $g\, a = f$
- lambda terms $\lambda y.\, y\, a$

Higher-Order Logic (HOL)

- functional variables $y\,a = g\,a\,b$
- partially applied functions $g\,a = f$
- lambda terms $\lambda y.\, y\,a$
- Booleans as terms $\lambda xy.\, P\,y \lor x$

SMT for HOL

Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,

SMT for HOL

Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,

SMT for HOL

Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,
- functional code.

SMT for HOL

Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,
- functional code.

SMT for HOL

Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,
- functional code.

HOL encoded in first-order logic

SMT for HOL

Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,
- functional code.

HOL encoded in first-order logic $\equiv$ structure loss

SMT for HOL

Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,
- functional code.

HOL encoded in first-order logic $\equiv$ structure loss $\approxeq$ performance loss

SMT for HOL

Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,
- functional code.

HOL encoded in first-order logic $\equiv$ structure loss $\approxeq$ performance loss

To work in HOL, both Input language and solver must be adapted!

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms
- dependent types

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms
- dependent types

SMTlib 2.7: selected features (lambdas, functional variables). To appear soon!

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms
- dependent types

SMTlib 2.7: selected features (lambdas, functional variables). To appear soon!

Already available in cvc5 (in 2.6)

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms
- dependent types

SMTlib 2.7: selected features (lambdas, functional variables). To appear soon!

Already available in cvc5 (in 2.6) with a minor setting change:

```
(set-logic QF_UFLRA)
(declare-const a Int)
(declare-fun g Int Int)
(declare-fun f (Int Int) Int)
(assert (forall ((x Int)) (= (g x) (f a x))))
(check-sat)
```

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms
- dependent types

SMTlib 2.7: selected features (lambdas, functional variables). To appear soon!

Already available in cvc5 (in 2.6) with a minor setting change:

```
(set-logic HO_QF_UFLRA)
(declare-const a Int)
(declare-fun g Int Int)
(declare-fun f (Int Int) Int)
(assert (forall ((x Int)) (= (g x) (f a x))))
(check-sat)
```

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms
- dependent types

SMTlib 2.7: selected features (lambdas, functional variables). To appear soon!

Already available in cvc5 (in 2.6) with a minor setting change:

```
(set-logic HO_ALL)
(declare-const a Int)
(declare-fun g Int Int)
(declare-fun f (Int Int) Int)
(assert (forall ((x Int)) (= (g x) (f a x))))
(check-sat)
```

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms
- dependent types

SMTlib 2.7: selected features (lambdas, functional variables). To appear soon!

Already available in cvc5 (in 2.6) with a minor setting change:

```
(set-logic HO_ALL)
(declare-const a Int)
(declare-const g (-> Int Int))
(declare-fun f (Int Int) Int)
(assert (forall ((x Int)) (= (g x) (f a x))))
(check-sat)
```

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms
- dependent types

SMTlib 2.7: selected features (lambdas, functional variables). To appear soon!

Already available in cvc5 (in 2.6) with a minor setting change:

```
(set-logic HO_ALL)
(declare-const a Int)
(declare-const g (-> Int Int))
(declare-fun f (Int Int) Int)
(assert (= g (f a)))
(check-sat)
```

SMTlib for HOL

SMTlib is being entirely redesigned for higher-order (and beyond) in the v3, featuring

- functional variables, partial applications, lambda terms, Boolean terms
- dependent types

SMTlib 2.7: selected features (lambdas, functional variables). To appear soon!

Already available in cvc5 (in 2.6) with a minor setting change:

```
(set-logic HO_ALL)
(declare-const a Int)
(declare-const g (-> Int Int))
(declare-fun f (Int Int) Int)
(assert (= g (lambda ((x Int)) (f x a))))
(check-sat)
```

HO-SMT solvers

Two main approaches to HO-SMT:

FOL to HOL

HOL to FOL

HO-SMT solvers

Two main approaches to HO-SMT:

FOL to HOL	datastructures lifting (heavy)
HOL to FOL	

HO-SMT solvers

Two main approaches to HO-SMT:

FOL to HOL	datastructures lifting (heavy)
HOL to FOL	encodings (light)

HO-SMT solvers

Two main approaches to HO-SMT:

veriT (light)	FOL to HOL	datastructures lifting (heavy)
	HOL to FOL	encodings (light)

HO-SMT solvers

Two main approaches to HO-SMT:

veriT (light)	FOL to HOL	datastructures lifting (heavy)
cvc4/cvc5 (heavy)	HOL to FOL	encodings (light)

HO-SMT solvers

Two main approaches to HO-SMT:

veriT (light)	FOL to HOL	datastructures lifting (heavy)
cvc4/cvc5 (heavy)	HOL to FOL	encodings (light)

What about instantiation?

HO-SMT solvers

Two main approaches to HO-SMT:

veriT (light)	FOL to HOL	datastructures lifting (heavy)
cvc4/cvc5 (heavy)	HOL to FOL	encodings (light)

What about instantiation?

trigger-based	conflict-based	model-based	enumerative

HO-SMT solvers

Two main approaches to HO-SMT:

veriT (light)	FOL to HOL	datastructures lifting (heavy)
cvc4/cvc5 (heavy)	HOL to FOL	encodings (light)

What about instantiation?

trigger-based	conflict-based	model-based	enumerative
○			

HO-SMT solvers

Two main approaches to HO-SMT:

veriT (light)	FOL to HOL	datastructures lifting (heavy)
cvc4/cvc5 (heavy)	HOL to FOL	encodings (light)

What about instantiation?

trigger-based	conflict-based	model-based	enumerative
○			×

HO-SMT solvers

Two main approaches to HO-SMT:

veriT (light)	FOL to HOL	datastructures lifting (heavy)
cvc4/cvc5 (heavy)	HOL to FOL	encodings (light)

What about instantiation?

trigger-based	conflict-based	model-based	enumerative
○		⟳	×

HO-SMT solvers

Two main approaches to HO-SMT:

veriT (light)	FOL to HOL	datastructures lifting (heavy)
cvc4/cvc5 (heavy)	HOL to FOL	encodings (light)

What about instantiation?

trigger-based	conflict-based	model-based	enumerative
○	↻	↻	×

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.
- If successful, build the instance from the model.

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.
- If successful, build the instance from the model.

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.
- If successful, build the instance from the model.

Current status:

- ○ theory

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.
- If successful, build the instance from the model.

Current status:

- ○ theory
- ⟳ Isabelle/HOL verification

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.
- If successful, build the instance from the model.

Current status:

- ○ theory
- ⟳ Isabelle/HOL verification
- ○ pseudo-code

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.
- If successful, build the instance from the model.

Current status:

- ○ theory
- ⟳ Isabelle/HOL verification
- ○ pseudo-code
- ○ core implementation (encoding, call to SAT)

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.
- If successful, build the instance from the model.

Current status:

- ○ theory
- ⟳ Isabelle/HOL verification
- ○ pseudo-code
- ○ core implementation (encoding, call to SAT)
- × full implementation (preprocessing, integration)

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.
- If successful, build the instance from the model.

Current status:

- ○ theory
- ⟳ Isabelle/HOL verification
- ○ pseudo-code
- ○ core implementation (encoding, call to SAT)
- × full implementation (preprocessing, integration)

Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.
- Apply SAT solving to find a model.
- If successful, build the instance from the model.

Current status:

- ○ theory
- ⟳ Isabelle/HOL verification
- ○ pseudo-code
- ○ core implementation (encoding, call to SAT)
- × full implementation (preprocessing, integration)

We want a new HOSMT solver first!

A Modular SMT Solver for Higher-Order

No good research vessel:

- veriT: light but code rot

A Modular SMT Solver for Higher-Order

No good research vessel:

- veriT: light but code rot
- cvc5: heavy, very high entry cost

A Modular SMT Solver for Higher-Order

No good research vessel:

- veriT: light but code rot
- cvc5: heavy, very high entry cost

A Modular SMT Solver for Higher-Order

No good research vessel:

- veriT: light but code rot
- cvc5: heavy, very high entry cost

We will create ModulariT, a new SMT solver for research in FOL and HOL.

A Modular SMT Solver for Higher-Order

No good research vessel:

- veriT: light but code rot
- cvc5: heavy, very high entry cost

We will create ModulariT, a new SMT solver for research in FOL and HOL.

Principles:

- Never sacrifice modularity for efficiency, to help research.

A Modular SMT Solver for Higher-Order

No good research vessel:

- veriT: light but code rot
- cvc5: heavy, very high entry cost

We will create ModulariT, a new SMT solver for research in FOL and HOL.

Principles:

- Never sacrifice modularity for efficiency, to help research.
- Gracefully lift first-order SMT to higher-order.

A Modular SMT Solver for Higher-Order

No good research vessel:

- veriT: light but code rot
- cvc5: heavy, very high entry cost

We will create ModulariT, a new SMT solver for research in FOL and HOL.

Principles:

- Never sacrifice modularity for efficiency, to help research.
- Gracefully lift first-order SMT to higher-order.
- Stay low level (C++) for efficiency and compatibility with other solvers (Z3, cvc5, bitwuzla, SPASS-SAT...)

To Conclude

SMT solving is going higher and faster!

To Conclude

SMT solving is going higher and faster!

- you can start playing with HOL in cvc5, but...

To Conclude

SMT solving is going higher and faster!

- you can start playing with HOL in cvc5, but...
- be patient for mature tools, or...

To Conclude

SMT solving is going higher and faster!

- you can start playing with HOL in cvc5, but...
- be patient for mature tools, or...
- try other higher-order tools

To Conclude

SMT solving is going higher and faster!

- you can start playing with HOL in cvc5, but...
- be patient for mature tools, or...
- try other higher-order tools

To Conclude

SMT solving is going higher and faster!

- you can start playing with HOL in cvc5, but...
- be patient for mature tools, or...
- try other higher-order tools (if you don't need arithmetic),

To Conclude

SMT solving is going higher and faster!

- you can start playing with HOL in cvc5, but...
- be patient for mature tools, or...
- try other higher-order tools (if you don't need arithmetic), e.g., Zipperposition, E, Vampire, Leo III, Lash...

To Conclude

SMT solving is going higher and faster!

- you can start playing with HOL in cvc5, but...
- be patient for mature tools, or...
- try other higher-order tools (if you don't need arithmetic), e.g., Zipperposition, E, Vampire, Leo III, Lash... and most importantly
- if you have ideas of new applications for HOSMT, let me know!

To Conclude

SMT solving is going higher and faster!

- you can start playing with HOL in cvc5, but...
- be patient for mature tools, or...
- try other higher-order tools (if you don't need arithmetic), e.g., Zipperposition, E, Vampire, Leo III, Lash... and most importantly
- if you have ideas of new applications for HOSMT, let me know!

To Conclude

SMT solving is going higher and faster!

- you can start playing with HOL in cvc5, but...
- be patient for mature tools, or...
- try other higher-order tools (if you don't need arithmetic), e.g., Zipperposition, E, Vampire, Leo III, Lash... and most importantly
- if you have ideas of new applications for HOSMT, let me know!

Looking forward to (future) HOSMT users!