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

Sophie Tourret
Journées Nationales du GDR GPL & AFADL
June 2024
Z3, Alt-Ergo, cvc5, ...
Z3, Alt-Ergo, cvc5, ...

SMT is well-known as a backend for many techniques, including:
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
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
SMT stands for Satisfiability Modulo Theories
SMT stands for Satisfiability Modulo Theories

An SMT solver determines the truth value of a formula.

A formula is ...
SMT stands for Satisfiability Modulo Theories

An SMT solver determines the truth value of a formula.

A formula is …

valid when always true,
SMT stands for Satisfiability Modulo Theories

An SMT solver determines the truth value of a formula.

A formula is . . .

valid when always true, 
satisfiable when true at least once,
SMT stands for Satisfiability Modulo Theories

An SMT solver determines the truth value of a formula.

A formula is . . .

valid when always true,

satisfiable when true at least once,

unsatisfiable when never true.
SMT solvers usually operate in first-order logic

+ interpreted symbols in given theories
SMT solvers usually operate in first-order logic

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

+ interpreted symbols in given theories
The Bases (2/2)

SMT solvers usually operate in first-order logic

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

+ interpreted symbols in given theories
  - $+$, $\times$, $\leq$, $=$, $\ldots$
SMT solvers usually operate in first-order logic

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

+ interpreted symbols in given theories
  - $+, \times, \leq, =, \ldots$

Example

$$a \leq b \land b \leq a + c \land c = 0 \land [a \neq b \lor (q(a) \land \neg q(f(b) + c))]$$
Inside an SMT solver

SMT formula

SMT solver
Returning to our example:

\[ a \leq b \land b \leq a + c \land c = 0 \land [a \neq b \lor (q(a) \land \neg q(f(b) + c))] \]
Returning to our example:

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

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)
(exit)
```
Returning to our example:

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

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)
(unsat)
```
Returning to our example:

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

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)
(exit)
```
Returning to our example:

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

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)
(exit)
```
Returning to our example:

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

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)
```
Returning to our example:

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

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)
(exit)
```
Returning to our example:

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

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)
(exit)
```
Returning to our example:

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

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)
    (exit)
```
Returning to our example:

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

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)
(exit)
```
Returning to our example:

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

encoded in SMT-LIB 2.0 format:

```lisp
(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)
(exit)
```
Returning to our example:

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

encoded in SMT-LIB 2.0 format:

```lisp
(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 (\leq a b) (\leq b (+ a c)) (= c 0)
           (or (\not (= a b))
                (and (q a) (\not (q (+ (f b) c)))))))
(check-sat)
(exit)
```
Returning to our example:

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

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)
(exit)
```
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)
(exit)
```
Inside an SMT solver

SMT solver

SMT formula
Inside an SMT solver

SMT formula

SMT solver

SAT solver
SAT Solving

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

Many uses:

- for cryptography
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
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
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
An SMT formula, e.g., our running example

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

cannot be handled by a SAT solver.
An SMT formula, e.g., our running example

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

cannot be handled by a SAT solver. It must be \textit{abstracted}, e.g.,

\[ a \leq b \land b \leq a + c \land c = 0 \land [a \neq b \lor (q(a) \land \neg q(f(b) + c))] \]
An SMT formula, e.g., our running example

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

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

\[ P \land b \leq a + c \land c = 0 \land [a \neq b \lor (q(a) \land \neg q(f(b) + c))] \]
An SMT formula, e.g., our running example

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

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

\[ P \land Q \land c = 0 \land [a \neq b \lor (q(a) \land \neg q(f(b) + c))] \]
An SMT formula, e.g., our running example

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

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

\[ P \land Q \land R \land [a \neq b \lor (q(a) \land \neg q(f(b) + c))] \]
An SMT formula, e.g., our running example

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

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

\[ P \land Q \land R \land [\neg S \lor (q(a) \land \neg q(f(b) + c))] \]
SAT Solving for SMT

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

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

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

\[ P \land Q \land R \land [ \neg S \lor ( T \land \neg q(f(b) + c))] \]
An SMT formula, e.g., our running example

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

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

\[ P \land Q \land R \land [ \neg S \lor (T \land \neg U)] \]
An SMT formula, e.g., our running example

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

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

\[ P \land Q \land R \land [ \neg S \lor ( T \land \neg U ) ] \]

If the abstracted formula is UNSAT, so is the SMT formula.
SAT Solving for SMT

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

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

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

\[ P \land Q \land R \land [\neg S \lor (T \land \neg U)] \]

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

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

\[ P \land Q \land R \land \neg S \]
Inside an SMT solver

SMT formula

SMT solver

SAT solver
Inside an SMT solver

SMT formula

SMT solver

SAT solver

Boolean Model
Inside an SMT solver

SMT formula

SMT solver

Theory reasoner

SAT solver

Boolean Model
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
- Non-linear arithmetic CAD, Gröbner bases...
  \[ x^2 + 2x - 8 = 0 \]

Data structures:
- Arrays uninterpreted symbols \( \text{read}(a, i) = b \)
- Bitvectors bit-blasting \( \text{concat} \ (b_{\text{v}}^i, b_{\text{v}}^j) = b_{\text{v}}^m \)
- Strings SAT + arithmetic \( "a" \cdot "bc" = "ab" \cdot "c" \)
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 \( 3x^2 + 2x - 8 = 0 \)
- Arrays uninterpreted symbols \( \text{read}(a, i) = b \)
- Bitvectors bit-blasting concatenation \( \text{bv}_i \text{bv}_j = \text{bv}_m \)
- Strings SAT + arithmetic \( \text{"a"} \cdot \text{"bc"} = \text{"ab"} \cdot \text{"c"} \)
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, Gr"obner bases... \( 3x^2 + 2x - 8 = 0 \)

Data structures:
arrays uninterpreted symbols read(a,i) = b
bitvectors bit-blasting concat \( bv_i \ \text{bv}_j = \text{bv}_m \)
strings SAT + arithmetic “a” · “bc” = “ab” · “c”
Theories for SMT

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

\[ P \land Q \land R \land \neg S \]

for our running example, it means

\[ a \leq b \land b \leq a + c \land c = 0 \land a \neq b. \]
Theories for SMT

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

\[ P \land Q \land R \land \neg S \]

for our running example, it means

\[ a \leq b \land b \leq a + c \land c = 0 \land a \neq b. \]

Then an LIA solver finds that both \( a = b \) and \( a \neq b \) must hold and returns false.
Theory solvers detect problematic assignments done by the SAT solver, e.g., if the SAT solver found the model

\[ P \land Q \land R \land \neg S \]

for our running example, it means

\[ a \leq b \land b \leq a + c \land c = 0 \land a \neq b. \]

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

The formula \( \neg P \lor \neg Q \lor \neg R \lor S \) is added to the abstracted formula before calling the SAT solver once more.
Inside an SMT solver

SMT formula

SMT solver

Theory reasoner

SAT solver

Boolean Model
Inside an SMT solver

SMT formula

SMT solver

Theory reasoner

SAT solver

Conflict clause

Boolean Model
Inside an SMT solver

SMT formula

SMT solver

Theory reasoner

Decision Procedure 1
Decision Procedure 2
...
Decision Procedure n
Combining Theories

If our example,

\[ P \land Q \land R \land \neg S \]

means in fact

\[ a \leq b \land b \leq a + c \land c = 0 \land f(a) \neq f(b). \]
If our example, 

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

means in fact

$$a \leq b \land b \leq a + c \land c = 0 \land f(a) \neq f(b).$$
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?
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: $a \leq b$, $b \leq a + c$, $c = 0$
- 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: \( a \leq b, b \leq a + c, c = 0 \implies b \leq a \)
- 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: $$ a \leq b, \ b \leq a + c, \ c = 0 \ \Rightarrow \ b \leq a \ \Rightarrow \ a = b $$
- 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**: \[ a \leq b, b \leq a + c, c = 0 \implies b \leq a \implies a = b \]
- **EUF**: \[ f(a) \neq f(b) \]
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:** \( a \leq b, b \leq a + c, c = 0 \implies b \leq a \implies a = b \)
- **EUF:** \( f(a) \neq f(b), a = b \)
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:** \( a \leq b, b \leq a + c, c = 0 \implies b \leq a \implies a = b \)
- **EUF:** \( f(a) \neq f(b), a = b \implies a \neq b \)
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**: \( a \leq b, b \leq a + c, c = 0 \Rightarrow b \leq a \Rightarrow a = b \)
- **EUF**: \( f(a) \neq f(b), a = b \Rightarrow a \neq b \Rightarrow \text{contradiction!} \)
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: \( a \leq b, \ b \leq a + c, \ c = 0 \implies b \leq a \implies a = b \)
- EUF: \( f(a) \neq f(b), \ a = b \implies a \neq b \implies \text{contradiction!} \)

Various techniques: Nelson-Open, Shostak, Gentleness, Politeness, ...
Inside an SMT solver

SMT formula

SMT solver

Conflict clause

Theory reasoner

SAT solver

Boolean Model
Inside an SMT solver

SMT formula

SMT solver

Quantifier-free SMT solver

Conflict clause

Theory reasoner

SAT solver

Boolean Model
Let us add to our improved running example,

\[ a \leq b \land b \leq a + c \land c = 0 \land [f(a) \neq f(b) \lor (q(a) \land \lnot q(f(b) + c))] \]

the quantified formula

\[ \forall x, y. (q(y) \implies q(g(y) + x)) \]
Let us add to our improved running example,
\[ a \leq b \land b \leq a + c \land c = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(f(b) + c))] \]
the quantified formula
\[ \forall x, y. (q(y) \implies q(g(y) + x)) \]
First the ground SMT solver will be queried for a model
Inside an SMT solver

SMT formula

SMT solver

Quantifier-free SMT solver

Conflict clause

Theory reasoner

SAT solver

Boolean Model

Model
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.
If our running example,

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

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

\[ a \leq b, b \leq a + c, c = 0, q(a), \neg q(f(b) + c) \]
If our running example,
\[
 a \leq b \land b \leq a + c \land c = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(f(b) + c))]
\]
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

SMT formula

SMT solver

Instantiation module

Instance

Model

Quantifier-free SMT solver

Conflict clause

Theory reasoner

Model

SAT solver

Boolean Model
for \[ a \leq b \land b \leq a + c \land c = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(f(b) + c))] \land \forall x, y. (q(y) \Rightarrow q(f(y) + x)) \]
given the model \( a \leq b, b \leq a + c, c = 0, q(a), \neg q(g(b) + c) \)
for \( a \leq b \land b \leq a + c \land c = 0 \land \left[ f(a) \neq f(b) \lor (q(a) \land \neg q(f(b) + c)) \right] \)
\[ \forall x, y. (q(y) \implies q(f(y) + x)) \]
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 \[ a \leq b \land b \leq a + c \land c = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(f(b) + c))] \]
\[ \forall x, y. (q(y) \implies q(f(y) + x)) \]

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.,
\[ q(a) \implies q(g(a) + f(b) - g(a)) \]

leads to a contradiction at the ground level!
Instantiation Techniques

There is no panacea!
There is no panacea!

Instantiation techniques:

- trigger-based

- conflict-based

- model-based

- enumerative
There is no panacea!

Instantiation techniques:

- trigger-based heuristic, to find unsat
- conflict-based heuristic, to find unsat
- model-based complete for decidable fragments, to find sat
- enumerative complete for finitely populated types
There is no panacea!

Instantiation techniques:

- trigger-based heuristic, to find unsat
- conflict-based
There is no panacea!

Instantiation techniques:

- trigger-based heuristic, to find unsat
- conflict-based also heuristic, to find unsat
There is no panacea!

Instantiation techniques:

- trigger-based heuristic, to find unsat
- conflict-based also heuristic, to find unsat, very efficient when it works
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
There is no panacea!

Instantiation techniques:

- trigger-based: heuristic, to find \textit{unsat}
- conflict-based: also heuristic, to find \textit{unsat}, very efficient when it works
- model-based: \textit{complete} for decidable fragments, to find \textit{sat}
- enumerative: \textit{complete} for finitely populated types
Inside an SMT solver

SMT formula

SMT solver

Instantiation module

Quantifier-free SMT solver

Theory reasoner

SAT solver

Conflict clause

Boolean Model

Model

Instance

UNSAT (proof/core)

Model
SMT Solving in Higher-Order Logic
Higher-Order Logic (HOL)

- functional variables $y a = g a b$

- partially applied functions $g a$

- lambda terms $\lambda y. y a$

- Booleans as terms $\lambda xy. P y \lor x$
Higher-Order Logic (HOL)

- functional variables $y \ a = g \ a \ b$
- partially applied functions $g \ a = f$
• 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 $\lambda xy. \ P y \lor x$
Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,
Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,
- functional code.
Higher-Order Logic is closer than First-Order Logic to:

- native language of proof assistants,
- theories like sets, streams, fixpoints, etc,
- functional code.
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
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
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 $\simeq$ performance loss
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 $\approx$ performance loss

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

```lisp
(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 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 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 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)
Two main approaches to HO-SMT:

- FOL to HOL
- HOL to FOL
Two main approaches to HO-SMT:

- FOL to HOL: datastructures lifting (heavy)
- HOL to FOL
Two main approaches to HO-SMT:

- FOL to HOL: datastructures lifting (heavy)
- HOL to FOL: encodings (light)
Two main approaches to HO-SMT:

- veriT (light)
- FOL to HOL
- data structures lifting (heavy)
- HOL to FOL
- encodings (light)
Two main approaches to HO-SMT:

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

<table>
<thead>
<tr>
<th>trigger-based</th>
<th>conflict-based</th>
<th>model-based</th>
<th>enumerative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>trigger-based</th>
<th>conflict-based</th>
<th>model-based</th>
<th>enumerative</th>
</tr>
</thead>
<tbody>
<tr>
<td>○</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>trigger-based</th>
<th>conflict-based</th>
<th>model-based</th>
<th>enumerative</th>
</tr>
</thead>
<tbody>
<tr>
<td>○</td>
<td></td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>trigger-based</th>
<th>conflict-based</th>
<th>model-based</th>
<th>enumerative</th>
</tr>
</thead>
<tbody>
<tr>
<td>○</td>
<td>○</td>
<td>●</td>
<td>×</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>trigger-based</th>
<th>conflict-based</th>
<th>model-based</th>
<th>enumerative</th>
</tr>
</thead>
<tbody>
<tr>
<td>○</td>
<td>✓</td>
<td>✔</td>
<td>×</td>
</tr>
</tbody>
</table>
Conflict-based Instantiation for HOSMT

- Encode the problem as a propositional constraints.

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!
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
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.
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.
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!

• 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!
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!
To Conclude

SMT solving is going higher and faster!

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

and most importantly

- if you have ideas of new applications for HOSMT, let me know!

Looking forward to (future) HOSMT users!
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

and most importantly
- if you have ideas of new applications for HOSMT, let me know!

Looking forward to (future) HOSMT users!
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),

and most importantly

- if you have ideas of new applications for HOSMT, let me know!

Looking forward to (future) HOSMT users!
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!
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!
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!