SMT: quantifiers, and future prospects

Pascal Fontaine (University of Liège)

Based on joint works with many, including
Haniel Barbosa, Jasmin Blanchette, Daniel El Ouaroue, Mathias Fleury, Mikolás Janota, Cezary Kaliszyk, Andrew Reynolds, Hans-Jörg Schurr, Sophie Tourret...

...and built on the work of many others (see citations)

IPAM Workshop, UCLA, February 2023
Outline

Introduction
Quantifiers and SMT: the basics
Instantiation techniques
Conclusion
References
Outline

Introduction
Quantifiers and SMT: the basics
Instantiation techniques
Conclusion
References
Motivation

- Formal proofs should not be mostly about proving easy things
- Automated theorem provers (ATPs) should prove the easy things for you
- ATP proofs can be replayed: confidence is not compromised
- E.g. Sledgehammer

- Proof obligations often use quantifiers
SMT = SAT + expressiveness

- SAT solvers
  \[ \neg[(p \Rightarrow q) \Rightarrow (\neg p \Rightarrow q) \Rightarrow q]] \]
- Congruence closure (uninterpreted symbols + equality)
  \[ a = b \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b))] \]
- and with arithmetic
  \[ a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))] \]
- ... 
- What about quantifiers?
Quantifiers in SMT

- Full first-order logic is undecidable
- First-order logic is semi-decidable
  refutationally complete procedures terminate on UNSAT
- If finite model property, then decidable
- Presburger with even one unary predicate is not even semi-decidable [Halper91]
- Pragmatic approaches are quite successful

Why does the pragmatic SMT approach work?
- Verification problems are big and shallow
- SMT appropriate for long, mostly ground, uninterpreted function reasoning

Working hypothesis

Quantifier handling for pure FOL will work most of the time sufficiently for SMT
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
  E-matching/trigger-based instantiation (e)
  Conflict-based instantiation (c)
  Model-based instantiation (m)
  Enumerative instantiation (u)
  Experimental evaluation

Conclusion

References
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
  - E-matching/trigger-based instantiation (e)
  - Conflict-based instantiation (c)
  - Model-based instantiation (m)
  - Enumerative instantiation (u)
  - Experimental evaluation

Conclusion

References
Unlike superposition-based FOL provers, SMT solvers essentially based on instantiation.

Herbrand instance of a Skolem formula $\forall \bar{x} \varphi(\bar{x})$: any ground formula $\varphi(\bar{t})$, where $\bar{t}$ are terms in the language.

**Theorem** (Herbrand)

A finite set of Skolem formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of Herbrand instances.

Caveats

- there should be at least one constant available for every sort
- holds for pure FOL, might not in presence of theories
Example

Is this syllogism correct?

All humans are mortal
All Greeks are humans

Then all Greeks are mortal

Artistotle
384–322 BC
Example

Is this syllogism correct?
All humans are mortal
All Greeks are humans
Then all Greeks are mortal

Translate to FOL

∀x. H(x) ⇒ M(x)
∀x. G(x) ⇒ H(x)
∀x. G(x) ⇒ M(x)

Checking the validity of this formula

∀x. H(x) ⇒ M(x)
∧ ∀x. G(x) ⇒ H(x)
⇒ ∀x. G(x) ⇒ M(x)

Checking the unsatisfiability of

∀x. H(x) ⇒ M(x), ∀x. G(x) ⇒ H(x), ¬∀x. G(x) ⇒ M(x)

Skolemize

∀x. H(x) ⇒ M(x), ∀x. G(x) ⇒ H(x), ¬G(s) ⇒ M(s)

Instantiate: add the two formulas (Herbrand instances)
H(s) ⇒ M(s), G(s) ⇒ H(s)

A ground (SAT/SMT) solver will deduce unsatisfiability.

Artistotle
384–322 BC
Example

Is this syllogism correct?
All humans are mortal
All Greeks are humans
Then all Greeks are mortal

Translate to FOL
\( \forall x. H(x) \Rightarrow M(x) \)
\( \forall x. G(x) \Rightarrow H(x) \)
\( \forall x. G(x) \Rightarrow M(x) \)

Artistotle
384–322 BC
Example

Is this syllogism correct?

All humans are mortal
All Greeks are humans
Then all Greeks are mortal

Translate to FOL

\[ \forall x. H(x) \Rightarrow M(x) \]
\[ \forall x. G(x) \Rightarrow H(x) \]
\[ \forall x. G(x) \Rightarrow M(x) \]

Checking the validity of this formula

\[ (\forall x. H(x) \Rightarrow M(x)) \land (\forall x. G(x) \Rightarrow H(x)) \Rightarrow \forall x. G(x) \Rightarrow M(x) \]

Artistotle
384–322 BC
Example

Is this syllogism correct?

All humans are mortal
All Greeks are humans
Then all Greeks are mortal

Translate to FOL

\[ \forall x. H(x) \Rightarrow M(x) \]
\[ \forall x. G(x) \Rightarrow H(x) \]
\[ \forall x. G(x) \Rightarrow M(x) \]

► Checking the validity of this formula

\[ \left( (\forall x. H(x) \Rightarrow M(x)) \land (\forall x. G(x) \Rightarrow H(x)) \right) \Rightarrow \forall x. G(x) \Rightarrow M(x) \]

► Checking the unsatisfiability of

\[ \forall x. H(x) \Rightarrow M(x), \forall x. G(x) \Rightarrow H(x), \neg \forall x. G(x) \Rightarrow M(x) \]
Example

Is this syllogism correct?

All humans are mortal
All Greeks are humans
Then all Greeks are mortal

Translate to FOL

\[ \forall x. H(x) \Rightarrow M(x) \]
\[ \forall x. G(x) \Rightarrow H(x) \]
\[ \forall x. G(x) \Rightarrow M(x) \]

Checking the validity of this formula

\[ \left( (\forall x. H(x) \Rightarrow M(x)) \land (\forall x. G(x) \Rightarrow H(x)) \right) \Rightarrow \forall x. G(x) \Rightarrow M(x) \]

Checking the unsatisfiability of

\[ \forall x. H(x) \Rightarrow M(x), \forall x. G(x) \Rightarrow H(x), \neg \forall x. G(x) \Rightarrow M(x) \]

Skolemize

\[ \forall x. H(x) \Rightarrow M(x), \forall x. G(x) \Rightarrow H(x), \neg (G(s) \Rightarrow M(s)) \]
Example

Is this syllogism correct?

All humans are mortal
All Greeks are humans
Then all Greeks are mortal

Translate to FOL

\[
\forall x. H(x) \Rightarrow M(x) \\
\forall x. G(x) \Rightarrow H(x) \\
\forall x. G(x) \Rightarrow M(x)
\]

Checking the validity of this formula

\[
\left( \left( \forall x. H(x) \Rightarrow M(x) \right) \land \left( \forall x. G(x) \Rightarrow H(x) \right) \right) \Rightarrow \forall x. G(x) \Rightarrow M(x)
\]

Checking the unsatisfiability of

\[
\forall x. H(x) \Rightarrow M(x), \forall x. G(x) \Rightarrow H(x), \neg \forall x. G(x) \Rightarrow M(x)
\]

Skolemize

\[
\forall x. H(x) \Rightarrow M(x), \forall x. G(x) \Rightarrow H(x), \neg (G(s) \Rightarrow M(s))
\]

Instantiate: add the two formulas (Herbrand instances)

\[
H(s) \Rightarrow M(s), G(s) \Rightarrow H(s)
\]
Example

Is this syllogism correct?

All humans are mortal
All Greeks are humans
Then all Greeks are mortal

Translate to FOL

\[ \forall x. H(x) \Rightarrow M(x) \]
\[ \forall x. G(x) \Rightarrow H(x) \]
\[ \forall x. G(x) \Rightarrow M(x) \]

▶ Checking the validity of this formula
\[ \left( \left( \forall x. H(x) \Rightarrow M(x) \right) \land \left( \forall x. G(x) \Rightarrow H(x) \right) \right) \Rightarrow \forall x. G(x) \Rightarrow M(x) \]

▶ Checking the unsatisfiability of
\[ \forall x. H(x) \Rightarrow M(x), \forall x. G(x) \Rightarrow H(x), \neg \forall x. G(x) \Rightarrow M(x) \]

▶ Skolemize
\[ \forall x. H(x) \Rightarrow M(x), \forall x. G(x) \Rightarrow H(x), \neg (G(s) \Rightarrow M(s)) \]

▶ Instantiate: add the two formulas (Herbrand instances)
\[ H(s) \Rightarrow M(s), G(s) \Rightarrow H(s) \]

▶ A ground (SAT/SMT) solver will deduce unsatisfiability.

Artistotle
384–322 BC
From SAT to SMT, ...

Input:  $a \leq b \land b \leq a + x \land x = 0 \land [f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))]$
From SAT to SMT,...

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

To SAT solver: \( p_{a \leq b} \land p_{b \leq a + x} \land p_{x = 0} \land [\neg p_{f(a) = f(b)} \lor (p_{q(a)} \land \neg p_{q(b + x)})] \)
From SAT to SMT,...

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

To SAT solver: \( p_{a \leq b} \land p_{b \leq a + x} \land p_{x=0} \land [\neg p_{f(a) = f(b)} \lor (p_{q(a)} \land \neg p_{q(b+x)})] \)

Boolean model: \( p_{a \leq b}, p_{b \leq a + x}, p_{x=0}, \neg p_{f(a) = f(b)} \)
From SAT to SMT,...

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

To SAT solver:  $p_{a \leq b} \land p_{b \leq a + x} \land p_{x = 0} \land [\neg p_{f(a) = f(b)} \lor (p_{q(a)} \land \neg p_{q(b + x)})]$

Boolean model:  $p_{a \leq b}, p_{b \leq a + x}, p_{x = 0}, \neg p_{f(a) = f(b)}$

Theory reasoner:  $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable
From SAT to SMT,

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

To SAT solver: \(p_{a \leq b} \land p_{b \leq a + x} \land p_{x=0} \land [\neg p_{f(a)=f(b)} \lor (p_{q(a)} \land \neg p_{q(b+x)})]\)

Boolean model: \(p_{a \leq b}, p_{b \leq a + x}, p_{x=0}, \neg p_{f(a)=f(b)}\)

Theory reasoner: \(a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)\) unsatisfiable

New clause: \(\neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x=0} \lor p_{f(a)=f(b)}\)

Conflict clauses are negation of unsatisfiable conjunctive sets of literals
From SAT to SMT, . . . and then to quantified SMT

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

To SAT solver: $p_{a \leq b} \land p_{b \leq a + x} \land p_{x = 0} \land [\neg p_{f(a) = f(b)} \lor (p_{q(a)} \land \neg p_{q(b + x)})]$

Boolean model: $p_{a \leq b}, p_{b \leq a + x}, p_{x = 0}, \neg p_{f(a) = f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause: $\neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x = 0} \lor p_{f(a) = f(b)}$

Conflict clauses are negation of unsatisfiable conjunctive sets of literals
From SAT to SMT, . . . and then to quantified SMT

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

To SAT solver: \( p_{a \leq b} \land p_{b \leq a+x} \land p_{x=0} \land [\neg p_{f(a)=f(b)} \lor (p_{q(a)} \land \neg p_{q(b+x)})] \)

Boolean model: \( p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)} \)

Theory reasoner: \( a \leq b, b \leq a + x, x = 0, f(a) \neq f(b) \) unsatisfiable

New clause: \( \neg p_{a \leq b} \lor \neg p_{b \leq a+x} \lor \neg p_{x=0} \lor p_{f(a)=f(b)} \)

Conflict clauses are negation of unsatisfiable conjunctive sets of literals
From SAT to SMT, . . . and then to quantified SMT

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

To SAT solver:  $p_{a \leq b} \land p_{b \leq a + x} \land p_{x = 0} \land \left[ \neg p_{f(a) = f(b)} \lor (p_{q(a)} \land \neg p_{q(b + x)}) \right]$

Boolean model:  $p_{a \leq b}, p_{b \leq a + x}, p_{x = 0}, \neg p_{f(a) = f(b)}$

Theory reasoner:  $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause:  $\neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x = 0} \lor p_{f(a) = f(b)}$

Conflict clauses are negation of unsatisfiable conjunctive sets of literals
From SAT to SMT,... and then to quantified SMT

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

To SAT solver: $p_{a \leq b} \land p_{b \leq a + x} \land p_{x = 0} \land [\neg p_{f(a) = f(b)} \lor (p_{q(a)} \land \neg p_{q(b + x)})]$

Boolean model: $p_{a \leq b}, p_{b \leq a + x}, p_{x = 0}, \neg p_{f(a) = f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause: $\neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x = 0} \lor p_{f(a) = f(b)}$

Conflict clauses are negation of unsatisfiable conjunctive sets of literals
From SAT to SMT, . . . and then to quantified SMT

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

To SAT solver: \( p_{a \leq b} \land p_{b \leq a + x} \land p_{x = 0} \land (\neg p_{f(a)} = f(b) \lor (p_{q(a)} \land \neg p_{q(b + x)})) \)

Boolean model: \( p_{a \leq b}, p_{b \leq a + x}, p_{x = 0}, \neg p_{f(a)} = f(b) \)

Theory reasoner: \( a \leq b, b \leq a + x, x = 0, f(a) \neq f(b) \) unsatisfiable

New clause: \( \neg p_{a \leq b} \lor \neg p_{b \leq a + x} \lor \neg p_{x = 0} \lor p_{f(a)} = f(b) \)

Conflict clauses are negation of unsatisfiable conjunctive sets of literals
From SAT to SMT, . . . and then to quantified SMT

- SMT formula
- SMT solver
  - Instance
  - Instantiation module
  - Ground SMT solver
    - Assignment
  - Model
  - UNSAT (proof/core)

Input:
\[ a \leq b \land b \leq a + x \land x = 0 \land \neg f(a) = f(b) \lor (q(a) \land \neg q(b + x)) \]

To SAT solver:
\[ \neg a \leq b \land \neg b \leq a + x \land \neg x = 0 \land \neg \neg f(a) = f(b) \lor (\neg q(a) \land q(b + x)) \]

Boolean model:
\[ \neg a \leq b, \neg b \leq a + x, \neg x = 0, \neg f(a) = f(b) \]

Theory reasoner:
\[ a \leq b, b \leq a + x, x = 0, f(a) \neq f(b) \text{ unsatisfiable} \]

New clause:
\[ \neg a \leq b \lor \neg b \leq a + x \lor \neg x = 0 \lor f(a) = f(b) \]

Conflict clauses are negation of unsatisfiable conjunctive sets of literals.
From SAT to SMT, . . . and then to quantified SMT
Input:
\( a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land \forall x. S(x) \equiv R(x) \)

To SAT solver:
\( p a = b \land p S(b) \land \neg p Q(a) \land \neg p R(a) \land p \forall x. S(x) \equiv R(x) \)

Boolean model:
\( p a = b, p S(b), \neg p Q(a), \neg p R(a), p \forall x. S(x) \equiv R(x) \)

Theory reasoner: fine! ... but does not understand \( \forall x. S(x) \equiv R(x) \)

Instantiation module: there is something to do with \( \forall x. S(x) \equiv R(x) \)

New clause:
\( \neg p a = b, \neg p S(b) \lor p R(a) \lor \neg p \forall x. S(x) \equiv R(x) \)

... too complicated to find/generate

What is the right formula to generate?
Input: \( a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge [\forall x \ Q(x) \vee \forall x . \ S(x) \equiv R(x)] \)
Input: $a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land \left[ \forall x Q(x) \lor \forall x . S(x) \equiv R(x) \right]$

To SAT solver: $p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land \left[ p_{\forall x Q(x)} \lor p_{\forall x . S(x) \equiv R(x)} \right]$
Instance?

\[
\text{Input: } a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \ Q(x) \lor \forall x . \ S(x) \equiv R(x)]
\]

\[
\text{To SAT solver: } p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x \ Q(x)} \lor p_{\forall x . \ S(x) \equiv R(x)}]
\]

\[
\text{Boolean model: } p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x . \ S(x) \equiv R(x)}
\]
Input: \( a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \ Q(x) \lor \forall x \ . \ S(x) \equiv R(x)] \)

To SAT solver: \( p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x \ Q(x)} \lor p_{\forall x \ . \ S(x)\equiv R(x)}] \)

Boolean model: \( p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \ . \ S(x)\equiv R(x)} \)

Theory reasoner: fine! . . . but does not understand \( \forall x \ . \ S(x) \equiv R(x) \)
Input: \( a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x Q(x) \lor \forall x . S(x) \equiv R(x)] \)

To SAT solver: \( p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x Q(x)} \lor p_{\forall x . S(x) \equiv R(x)}] \)

Boolean model: \( p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x . S(x) \equiv R(x)} \)

Theory reasoner: fine! . . . but does not understand \( \forall x . S(x) \equiv R(x) \)

Instantiation module: there is something to do with \( \forall x . S(x) \equiv R(x) \)
Input: $a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x Q(x) \lor \forall x . S(x) \equiv R(x)]$

To SAT solver: $p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x Q(x)} \lor p_{\forall x . S(x)\equiv R(x)}]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x . S(x)\equiv R(x)}$

Theory reasoner: fine! . . . but does not understand $\forall x . S(x) \equiv R(x)$

Instantiation module: there is something to do with $\forall x . S(x) \equiv R(x)$

New clause: $\neg p_{a=b}, \neg p_{S(b)} \lor p_{R(a)} \lor \neg p_{\forall x . S(x)\equiv R(x)}$
Input: $a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \ Q(x) \lor \forall x \ S(x) \equiv R(x)]$

To SAT solver: $p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x \ Q(x)} \lor p_{\forall x \ S(x)} \equiv R(x)]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \ S(x)} \equiv R(x)$

Theory reasoner: fine! . . . but does not understand $\forall x \ S(x) \equiv R(x)$

Instantiation module: there is something to do with $\forall x \ S(x) \equiv R(x)$

New clause: $\neg p_{a=b}, \neg p_{S(b)} \lor p_{R(a)} \lor \neg p_{\forall x \ S(x)} \equiv R(x)$

. . . too complicated to find/generate
Input: \( a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \ Q(x) \lor \forall x \ . \ S(x) \equiv R(x)] \)

To SAT solver: \( p_{a=b} \land p_{S(b)} \land p_Q(a) \land p_R(a) \land [p_{\forall x \ Q(x)} \lor p_{\forall x \ . \ S(x)\equiv R(x)}] \)

Boolean model: \( p_{a=b}, p_{S(b)}, \neg p_Q(a), \neg p_R(a), p_{\forall x \ . \ S(x)\equiv R(x)} \)

Theory reasoner: fine! . . . but does not understand \( \forall x \ . \ S(x) \equiv R(x) \)

Instantiation module: there is something to do with \( \forall x \ . \ S(x) \equiv R(x) \)

New clause: \( \neg p_{a=b}, \neg p_{S(b)} \lor p_R(a) \lor \neg p_{\forall x \ . \ S(x)\equiv R(x)} \)

. . . too complicated to find/generate

What is the right formula to generate?
Input: \( a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \ Q(x) \lor \forall x . \ S(x) \equiv R(x)] \)

To SAT solver: \( p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x \ Q(x)} \lor p_{\forall x . \ S(x) \equiv R(x)}] \)

Boolean model: \( p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \ S(x) \equiv R(x)} \)

Instantiation module: there is something to do with \( \forall x . \ S(x) \equiv R(x) \)

What is the right formula to generate?
Input: \( a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \ Q(x) \lor \forall x \ . \ S(x) \equiv R(x)] \)

To SAT solver: \( p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x \ Q(x)} \lor p_{\forall x \ . \ S(x)\equiv R(x)}] \)

Boolean model: \( p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \ . \ S(x)\equiv R(x)} \)

Instantiation module: there is something to do with \( \forall x \ . \ S(x) \equiv R(x) \)

What is the right formula to generate?
\( S(a) \equiv R(a) \) is not right
Input:  \( a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \ Q(x) \lor \forall x . \ S(x) \equiv R(x)] \)

To SAT solver:  \( p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x \ Q(x)} \lor p_{\forall x . \ S(x)\equiv R(x)}] \)

Boolean model:  \( p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x . \ S(x)\equiv R(x)} \)

Instantiation module: there is something to do with \( \forall x . \ S(x) \equiv R(x) \)

What is the right formula to generate?

\( S(a) \equiv R(a) \) is not right

We want \( S(a) \equiv R(a) \) whenever \( p_{\forall x . \ S(x)\equiv R(x)} \) is in the Boolean model
Instance?

Input: $a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \ Q(x) \lor \forall x \ . \ S(x) \equiv R(x)]$

To SAT solver: $p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x \ Q(x)} \lor p_{\forall x \ . \ S(x) \equiv R(x)}]$

Boolean model: $p_{a=b}, p_{S(b)} , \neg p_{Q(a)} , \neg p_{R(a)} , p_{\forall x \ . \ S(x) \equiv R(x)}$

Instantiation module: there is something to do with $\forall x . S(x) \equiv R(x)$

What is the right formula to generate?

$S(a) \equiv R(a)$ is not right

We want $S(a) \equiv R(a)$ whenever $p_{\forall x . S(x) \equiv R(x)}$ is in the Boolean model

$(\forall x . S(x) \equiv R(x)) \Rightarrow (S(a) \equiv R(a))$ would do
Input: \( a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \: Q(x) \lor \forall x . \: S(x) \equiv R(x)] \)

To SAT solver: \( p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x \: Q(x)} \lor p_{\forall x . \: S(x) \equiv R(x)}] \)

Boolean model: \( p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x . \: S(x) \equiv R(x)} \)

Instantiation module: there is something to do with \( \forall x . \: S(x) \equiv R(x) \)

What is the right formula to generate?

\( S(a) \equiv R(a) \) is not right

We want \( S(a) \equiv R(a) \) whenever \( p_{\forall x . \: S(x) \equiv R(x)} \) is in the Boolean model

\( (\forall x . \: S(x) \equiv R(x)) \Rightarrow (S(a) \equiv R(a)) \) would do

\( \neg p_{\forall x . \: S(x) \equiv R(x)} \lor (p_{S(a)} \equiv p_{R(a)}) \) at the propositional level
Input: \( a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land [\forall x \ Q(x) \lor \forall x \ . \ S(x) \equiv R(x)] \)

To SAT solver: \( p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land [p_{\forall x \ Q(x)} \lor p_{\forall x \ . \ S(x) \equiv R(x)}] \)

Boolean model: \( p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \ . \ S(x) \equiv R(x)} \)

Instantiation module: there is something to do with \( \forall x \ . \ S(x) \equiv R(x) \)

What is the right formula to generate?

\( S(a) \equiv R(a) \) is not right

We want \( S(a) \equiv R(a) \) whenever \( p_{\forall x \ . \ S(x) \equiv R(x)} \) is in the Boolean model

\( (\forall x \ . \ S(x) \equiv R(x)) \Rightarrow (S(a) \equiv R(a)) \) would do

\( \neg p_{\forall x \ . \ S(x) \equiv R(x)} \lor (p_{S(a)} \equiv p_{R(a)}) \) at the propositional level

Together with \( \forall x \ Q(x) \Rightarrow Q(a) \), this grounds the problem
Instance in an SMT context

\[ \forall \vec{x} \varphi(\vec{x}) \Rightarrow \varphi\sigma \]

where \( \sigma \) is a ground substitution for variables \( \vec{x} \)

E.g. \( \forall \vec{x} \varphi(\vec{x}) \) is \( \forall x . S(x) \equiv R(x) \), \( \sigma \) is \( x \mapsto a \), \( \varphi\sigma \) is \( S(a) \equiv R(a) \)

Remarks

- Above formula is a FOL tautology. E.g. \( (\forall x . S(x) \equiv R(x)) \Rightarrow (S(a) \equiv R(a)) \)
- \( \forall \vec{x} \varphi(\vec{x}) \) gets abstracted as a propositional variable in the SAT solver, that has a meaning only for the instantiation module
- \( \varphi\sigma \) gets abstracted as a Boolean combination of propositional variables... 
- ... that have meaning at the level of the \textit{ground} theory reasoner
- \( \varphi\sigma \) gets “activated”/relevant only in the models where \( p_{\forall \vec{x} \varphi(\vec{x})} \) is true.

We might refer to \( \varphi\sigma \) as the instance, but remember: all is fine at the level of the SAT solver/ground SMT solver
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
  - E-matching/trigger-based instantiation (e)
  - Conflict-based instantiation (c)
  - Model-based instantiation (m)
  - Enumerative instantiation (u)
  - Experimental evaluation

Conclusion

References
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
   E-matching/trigger-based instantiation (e)
   Conflict-based instantiation (c)
   Model-based instantiation (m)
   Enumerative instantiation (u)
   Experimental evaluation

Conclusion

References
Instantiation techniques

The framework

Ground SMT solver enumerates assignments $E \cup Q$

- $E$ set of ground literals
- $Q$ set of quantified clauses

Instantiation module generates instances of $Q$ that will further feed $E$

classic Herbrand Theorem: instantiate with all possible terms in language
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques

- E-matching/trigger-based instantiation (e)
- Conflict-based instantiation (c)
- Model-based instantiation (m)
- Enumerative instantiation (u)
  Experimental evaluation

Conclusion

References
E-matching/Trigger-based instantiation (e) [Detlefs05, deMoura07]

Search for relevant instances according to a set of triggers and $E$-matching

\[ E = \{ \neg P(a), \neg P(b), P(c), \neg R(b) \} \]
\[ Q = \{ \forall x. P(x) \lor R(x) \} \]

Assume trigger $P(x)$

Find substitution $\sigma$ for $x$ such that $P(x)$ is a known term (in $E$)

Three suitable substitutions:
- $x \mapsto a$
- $x \mapsto b$
- $x \mapsto c$

E.g.

$E | = P(x)[x/\sigma] = P(a)$ and $P(a) \in E$

Formally

\[ e(E, \forall \bar{x}. \phi) \]

1. Select a set of triggers $\{ \bar{t}_1, \ldots, \bar{t}_n \}$ for $\forall \bar{x}. \phi$
2. For each $i = 1, \ldots, n$, select a set of substitutions $S_i$ s.t. for each $\sigma \in S_i$, $E | = \bar{t}_i \sigma = \bar{g}_i$ for some tuple $\bar{g}_i \in T_E$.
3. Return $S_{i=1} S_i$
E-matching/Trigger-based instantiation (e) [Detlefs05, deMoura07]

Search for relevant instances according to a set of triggers and E-matching

- $E = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $Q = \{\forall x. P(x) \vee R(x)\}$

- Assume trigger $P(x)$

- Find substitution $\sigma$ for $x$ such $P(x)$ is a know term (in $E$)

- Three suitable substitutions: $x \leftarrow a$, $x \leftarrow b$, or $x \leftarrow c$
  E.g. $E \models P(x)[x/a] = P(a)$ and $P(a) \in E$

- Formally

\[
e(E, \forall \vec{x}. \varphi) = \begin{cases} 
1. \text{Select a set of triggers } \{\vec{t}_1, \ldots, \vec{t}_n\} \text{ for } \forall \vec{x}. \varphi \\
2. \text{For each } i = 1, \ldots, n, \text{ select a set of substitutions } S_i \text{ s.t. for each } \sigma \in S_i, E \models \vec{t}_i \sigma = \vec{g}_i \text{ for some tuple } \vec{g}_i \in \mathcal{T}_E. \\
3. \text{Return } \bigcup_{i=1}^{n} S_i
\end{cases}
\]
E-matching/Trigger-based instantiation

Ideal for expanding definitions/rewriting rules

▷ Example

\[ \forall x \forall y. \text{sister}(x, y) \equiv (\text{female}(x) \land \text{mother}(x) = \text{mother}(y) \land \text{father}(x) = \text{father}(y)) \]

\[
\text{sister}(\text{Eliane}, \text{Eloïse})
\]

\[
\text{sister}(\text{Eloïse}, \text{Elisabeth})
\]

\[
\neg\text{sister}(\text{Eliane}, \text{Elisabeth})
\]

▷ Choosing instantiation trigger \text{sister}(x, y) suffices for SMT solver to prove unsatisfiability

Remarks

▷ Decision procedure for, e.g., expressive arrays, lists [Dross16]

▷ Mostly efficient (see later evaluation)

▷ But can easily blow or miss the right instances

▷ Requires triggers (human or auto-generated)
Instantiation method issue: number of useless generated instances

It often occurs that > 99% of 100k generated instances are useless
E-matching/Trigger-based instantiation, prospects
Machine learning for instance filtering

- Instantiation method issue: number of useless generated instances
- It often occurs that > 99% of 100k generated instances are useless

An opportunity for machine learning

separate the wheat from the chaff: select the useful instances
E-matching/Trigger-based instantiation, prospects

Machine learning for instance filtering

- Instantiation method issue: number of useless generated instances
- It often occurs that $>99\%$ of 100k generated instances are useless

An opportunity for machine learning

separate the wheat from the chaff: select the useful instances

- We investigated XGBoost to filter instances [Blanchette19]
E-matching/Trigger-based instantiation, prospects
Machine learning for instance filtering

- Instantiation method issue: number of useless generated instances
- It often occurs that > 99% of 100k generated instances are useless

An opportunity for machine learning
separate the wheat from the chaff: select the useful instances

- We investigated XGBoost to filter instances [Blanchette19]
- Trained on successful proofs (good instance ← survives pruning of proof)
ML for instance filtering: experimental results

<table>
<thead>
<tr>
<th></th>
<th>30 s</th>
<th>60 s</th>
<th>120 s</th>
<th>180 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>veriT</td>
<td>2896</td>
<td>2913</td>
<td>2923</td>
<td>2929</td>
</tr>
<tr>
<td>veriT((M))</td>
<td>2907</td>
<td>2917</td>
<td>2925</td>
<td>2936</td>
</tr>
<tr>
<td>veriT((M^2))</td>
<td>2916</td>
<td>2927</td>
<td>2935</td>
<td>2944</td>
</tr>
<tr>
<td>veriT((M + M^2))</td>
<td>2936</td>
<td>2959</td>
<td>2969</td>
<td>2975</td>
</tr>
<tr>
<td>veriT + portfolio</td>
<td>3181</td>
<td>3215</td>
<td>3228</td>
<td>3234</td>
</tr>
<tr>
<td>veriT((M + M^2)) + portfolio</td>
<td>3190</td>
<td>3247</td>
<td>3312</td>
<td>3322</td>
</tr>
<tr>
<td>Vampire smtcomp mode</td>
<td>3154</td>
<td>3165</td>
<td>3175</td>
<td>3197</td>
</tr>
<tr>
<td>CVC4 portfolio</td>
<td>3311</td>
<td>3345</td>
<td>3393</td>
<td>3404</td>
</tr>
</tbody>
</table>

Results on the benchmarks in the UF category of the SMT-LIB

- veriT: vanilla
- veriT(\(M\)): veriT with instance selection trained with veriT successes
- veriT(\(M^2\)): veriT with instance selection trained with veriT(\(M\)) successes
- veriT(\(M + M^2\)): portfolio of above two
- veriT(\(M + M^2\)) + portfolio of several strategies, with instance selection
ML for instance filtering: number of instances on test + training set

veriT on UF SMT-LIB benchmarks (with vs. without filtering)
ML for instance filtering: number of instances on test set only

veriT on UF SMT-LIB benchmarks (with vs. without filtering)
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
  E-matching/trigger-based instantiation (e)
  Conflict-based instantiation (c)
  Model-based instantiation (m)
  Enumerative instantiation (u)
  Experimental evaluation

Conclusion

References
Conflict-based instantiation ($c$) [Reynolds14]

Search for one instance of one quantified formula in $Q$ that is unsatisfiable together with $E$
Conflict-based instantiation (c) [Reynolds14]

Search for one instance of one quantified formula in $Q$ that is unsatisfiable together with $E$

- $E = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $Q = \{\forall x. P(x) \lor R(x)\}$

- Since $E, P(b) \lor R(b) \models \bot$, this strategy returns $x \mapsto b$

- Formally

  $$c(E, \forall \bar{x}. \varphi) \quad \text{Either return } \sigma \text{ where } E \models \neg \varphi \sigma, \text{ or return } \emptyset$$
\( c: \) solving the problem

\[ E \land \psi \sigma \models \bot, \text{ for some } \forall \bar{x} \psi \in Q \]
c: solving the problem

\[ E \models \neg \psi \sigma, \text{ for some } \forall \bar{x} \psi \in Q \]
c: solving the problem

\[ E \models \neg \psi \sigma, \text{ for some } \forall \bar{x} \psi \in Q \]

\[ E = \{ f(a) = f(b), g(b) \neq h(c) \}, \quad Q = \{ \forall xyz. f(x) = f(z) \rightarrow h(y) = g(z) \} \]
c: solving the problem

\[ E \models \neg \psi \sigma, \text{ for some } \forall \bar{x} \psi \in Q \]

\[ E = \{ f(a) = f(b), g(b) \neq h(c) \}, \quad Q = \{ \forall xyz. f(x) = f(z) \rightarrow h(y) = g(z) \} \]

\[ f(a) = f(b) \land g(b) \neq h(c) \models (f(x) = f(z) \land h(y) \neq g(z)) \sigma \]
c: solving the problem

\[ E \models \neg \psi \sigma, \text{ for some } \forall \bar{x} \psi \in Q \]

\[ E = \{f(a) = f(b), g(b) \neq h(c)\}, \quad Q = \{\forall xyz. f(x) = f(z) \to h(y) = g(z)\} \]

\[ f(a) = f(b) \land g(b) \neq h(c) \models (f(x) = f(z) \land h(y) \neq g(z)) \sigma \]

- Each literal in the right hand side restricts \( \sigma \)
c: solving the problem

\[ E \models \neg \psi \sigma, \text{ for some } \forall \bar{x} \psi \in Q \]

\[ E = \{f(a) = f(b), \ g(b) \neq h(c)\}, \ Q = \{\forall xyz. \ f(x) = f(z) \rightarrow h(y) = g(z)\} \]

\[ f(a) = f(b) \land g(b) \neq h(c) \models (f(x) = f(z) \land h(y) \neq g(z)) \sigma \]

- Each literal in the right hand side restricts \( \sigma \)
  - \( f(x) = f(z) \): either \( x = z \) or \( x = a \land z = b \) or \( x = b \land z = a \)
c: solving the problem

\[
E \models \neg \psi \sigma, \text{ for some } \forall x \psi \in Q
\]

\[
E = \{f(a) = f(b), \ g(b) \neq h(c)\}, \ Q = \{\forall xyz. f(x) = f(z) \rightarrow h(y) = g(z)\}
\]

\[
f(a) = f(b) \land g(b) \neq h(c) \models (f(x) = f(z) \land h(y) \neq g(z)) \sigma
\]

- Each literal in the right hand side restricts \( \sigma \)
  - \( f(x) = f(z) \): either \( x = z \) or \( x = a \land z = b \) or \( x = b \land z = a \)
  - \( h(y) \neq g(z) \): \( y = c \land z = b \)
c: solving the problem

\[ E \models \neg \psi \sigma, \text{ for some } \forall \bar{x} \psi \in Q \]

\[ E = \{ f(a) = f(b), \ g(b) \neq h(c) \}, \ Q = \{ \forall xyz. \ f(x) = f(z) \rightarrow h(y) = g(z) \} \]

\[ f(a) = f(b) \land g(b) \neq h(c) \models (f(x) = f(z) \land h(y) \neq g(z)) \sigma \]

- Each literal in the right hand side restricts \( \sigma \)
  - \( f(x) = f(z) \): either \( x = z \) or \( x = a \land z = b \) or \( x = b \land z = a \)
  - \( h(y) \neq g(z) \): \( y = c \land z = b \)

\[ \sigma = \{ x \mapsto b, \ y \mapsto c, \ z \mapsto b \} \]
c: solving the problem

\[ E \models \neg \psi \sigma, \text{ for some } \forall \bar{x} \psi \in Q \]

\[ E = \{ f(a) = f(b), g(b) \neq h(c) \}, \quad Q = \{ \forall xyz. f(x) = f(z) \rightarrow h(y) = g(z) \} \]

\[ f(a) = f(b) \land g(b) \neq h(c) \models (f(x) = f(z) \land h(y) \neq g(z)) \sigma \]

- Each literal in the right hand side restricts \( \sigma \)
  - \( f(x) = f(z) \): either \( x = z \) or \( x = a \land z = b \) or \( x = b \land z = a \)
  - \( h(y) \neq g(z) \): \( y = c \land z = b \)

\[ \sigma = \{ x \mapsto b, \ y \mapsto c, \ z \mapsto b \} \]

or

\[ \sigma = \{ x \mapsto a, \ y \mapsto c, \ z \mapsto b \} \]
C: solving the problem

\[ E \models \neg \psi \sigma, \text{ for some } \forall x \psi \in Q \]

\[ E = \{ f(a) = f(b), g(b) \neq h(c) \}, \ Q = \{ \forall xyz. f(x) = f(z) \rightarrow h(y) = g(z) \} \]

\[ f(a) = f(b) \land g(b) \neq h(c) \models (f(x) = f(z) \land h(y) \neq g(z)) \sigma \]

- Each literal in the right hand side restricts \( \sigma \)
  - \( f(x) = f(z) \): either \( x = z \) or \( x = a \land z = b \) or \( x = b \land z = a \)
  - \( h(y) \neq g(z) \): \( y = c \land z = b \)

\[ \sigma = \{ x \mapsto b, \ y \mapsto c, \ z \mapsto b \} \]

or

\[ \sigma = \{ x \mapsto a, \ y \mapsto c, \ z \mapsto b \} \]
c: solving the problem with $E$-ground (dis)unification

Given conjunctive sets of equality literals $E$ and $L$, with $E$ ground, find substitution $\sigma$ s.t. $E \models L\sigma$
c: solving the problem with $E$-ground (dis)unification

Given conjunctive sets of equality literals $E$ and $L$, with $E$ ground, find substitution $\sigma$ s.t. $E \models L\sigma$

▶ Variant of classic (non-simultaneous) rigid $E$-unification
Given conjunctive sets of equality literals $E$ and $L$, with $E$ ground, find substitution $\sigma$ s.t. $E \models L\sigma$

- Variant of classic (non-simultaneous) rigid $E$-unification
- NP-complete
  - NP: solutions can be restricted to ground terms in $E \cup L$
  - NP-hard: reduction of 3-SAT
- CCFV: congruence closure with free variables [Barbosa17]
  - sound, complete and terminating calculus for solving $E$-ground (dis)unification
  - goal oriented
  - efficient in practice
- Still, 60% of time in veriT
Evaluation on SMT-LIB, TPTP, Isabelle benchmarks

Using conflict-based instantiation (cvc4+ci), require an order of magnitude fewer instances to prove unsatisfiability w.r.t. E-matching alone
veriT: + 800 out of 1785 unsolved problems
CVC4: + 200 out of 745 unsolved problems

* experiments in the "UF", "UFLIA", "UFLRA" and "UFIDL" categories of SMT-LIB, which have 10495 benchmarks annotated as unsatisfiable, with 30s timeout.
Conflicting instances, prospects

- Still, 60% of time in veriT
- CCFV is an NP-complete problem
- It can be encoded into SAT
- We expect careful encoding of CCFV into SAT will provide efficient procedure
- We are investigating a SAT-based algorithm for higher-order CCFV

- Conflicting instances only work for one instance
- Finding out a pair of instances that contradict a model?
- Maybe use superposition? Extend algorithm to find conflicts with several clauses?
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
  - E-matching/trigger-based instantiation (e)
  - Conflict-based instantiation (c)
  - Model-based instantiation (m)
  - Enumerative instantiation (u)
  - Experimental evaluation

Conclusion

References
Model-based instantiation/MBQI ($m$) [Ge09]

Build a candidate model for $E \cup Q$ and instantiate with counter-examples from model checking
Model-based instantiation/MBQI (m) [Ge09]

Build a candidate model for $E \cup Q$ and instantiate with counter-examples from model checking

- $E = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $Q = \{\forall x. P(x) \lor R(x)\}$

- Ground solver provides a partial model...

- ...extended to a full model s.t. $P^M = \lambda x. \text{ite}(x = c, \top, \bot)$ and $R^M = \lambda x. \bot$

- Since $M \models \neg (P(a) \lor R(a))$, this strategy may return $x \mapsto a$

- Formally

\[
\text{m}(E, \forall \vec{x}. \varphi) \quad \begin{align*}
1. & \quad \text{Construct a model } M \text{ for } E \\
2. & \quad \text{Return } \vec{x} \mapsto \bar{t} \text{ where } \bar{t} \in \mathcal{T}(E) \text{ and } M \models \neg \varphi[\vec{x}/\bar{t}], \\
& \quad \text{or } \emptyset \text{ if none exists}
\end{align*}
\]
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
  E-matching/trigger-based instantiation (e)
  Conflict-based instantiation (c)
  Model-based instantiation (m)
  Enumerative instantiation (u)

Experimental evaluation

Conclusion

References
Why can’t we directly use Herbrand instantiation?

**Theorem (Herbrand)**

*A finite set of Skolem formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of Herbrand instances*
Why can’t we directly use Herbrand instantiation?

**Theorem** (Herbrand)

A finite set of Skolem formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of Herbrand instances

- The earliest theorem provers relied on *Herbrand instantiation*
  - Instantiate with all possible terms in the language
- Enumerating all instances is unfeasible in practice!
- Enumerative instantiation was then discarded
Why can’t we directly use Herbrand instantiation?

**Theorem (Herbrand)**

A finite set of Skolem formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of Herbrand instances

- The earliest theorem provers relied on *Herbrand instantiation*
  - Instantiate with all possible terms in the language
- Enumerating all instances is unfeasible in practice!
- Enumerative instantiation was then discarded

Revisiting enumerative instantiation with benefits:
- strengthening of Herbrand theorem
- efficient implementation techniques
**Theorem** (Strengthened Herbrand)

If $R$ is a (possibly infinite) set of instances of $Q$ closed under $Q$-instantiation w.r.t. itself and if $E \cup R$ is satisfiable, then $E \cup Q$ is satisfiable.
**Theorem** (Strengthened Herbrand)

If there exists an infinite sequence of finite satisfiable sets of ground literals $E_i$ and of finite sets of ground instances $Q_i$ of $Q$ such that

- $Q_i = \{ \varphi \sigma \mid \forall \bar{x}. \varphi \in Q, \text{dom}(\sigma) = \{\bar{x}\} \land \text{ran}(\sigma) \subseteq T(E_i)\}$;
- $E_0 = E$, $E_{i+1} \models E_i \cup Q_i$;

then $E \cup Q$ is satisfiable in the empty theory with equality.
Theorem (Strengthened Herbrand)

If there exists an infinite sequence of finite satisfiable sets of ground literals $E_i$ and of finite sets of ground instances $Q_i$ of $Q$ such that

- $Q_i = \{ \varphi \sigma \mid \forall \overline{x}. \varphi \in Q, \text{dom}(\sigma) = \{\overline{x}\} \land \text{ran}(\sigma) \subseteq \mathcal{T}(E_i)\}$;
- $E_0 = E, E_{i+1} \models E_i \cup Q_i$;

then $E \cup Q$ is satisfiable in the empty theory with equality.

Direct application to

- Ground solver enumerates assignments $E \cup Q$
- Instantiation module generates instances of $Q$
Enumerative instantiation ($u$)

$u(E, \forall \bar{x}. \varphi)$

1. Choose an ordering $\preceq$ on tuples of ground terms
2. Return $\bar{x} \mapsto \bar{t}$ where $\bar{t}$ is a minimal tuple of terms w.r.t $\preceq$, such that $\bar{t} \in T(E)$ and $E \not\models \varphi[\bar{x}/\bar{t}]$, or $\emptyset$ if none exist

$\Rightarrow E = \{-P(a), \neg P(b), P(c), \neg R(b)\}$ and $Q = \{\forall x. P(x) \lor R(x)\}$

$\Rightarrow u$ chooses an ordering on tuples of terms, e.g. $a \prec b \prec c$

$\Rightarrow$ Since $E \not\models P(a) \lor R(a)$, enumerative instantiation returns $x \mapsto a$
u as an alternative for m

- Enumerative instantiation plays a similar role to m

- It can also serve as a “completeness fallback” to c and e

- However, u has advantages over m for UNSAT problems

- And it is significantly simpler to implement
  - no model building
  - no model checking
Example

\[ E = \{ \neg P(a), R(b), S(c) \} \]
\[ Q = \{ \forall x. R(x) \lor S(x), \forall x. \neg R(x) \lor P(x), \forall x. \neg S(x) \lor P(x) \} \]
\[ M = \left\{ \begin{array}{l}
P_M = \lambda x. \bot, \\
R_M = \lambda x. \text{ite}(x = b, \top, \bot), \\
S_M = \lambda x. \text{ite}(x = c, \top, \bot), \\
\end{array} \right\}, \quad a \prec b \prec c \]

\[
\begin{array}{c|c|c|c|c}
\varphi & \text{x s.t. } M \models \neg \varphi & \text{x s.t. } E \not\models \varphi & \text{m}(E, \forall x. \varphi) & \text{u}(E, \forall x. \varphi) \\
\hline
R(x) \lor S(x) & a & a & x \mapsto a & x \mapsto a \\
\neg R(x) \lor P(x) & b & a, b, c & x \mapsto b & x \mapsto a \\
\neg S(x) \lor P(x) & c & a, b, c & x \mapsto c & x \mapsto a \\
\end{array}
\]

- **u** instantiates uniformly so that less new terms are introduced
- **m** instantiates depending on how model was built
- **u** directly leads to \( E \land Q[x/a] \models \bot \)
Advanced \textbf{u}: restricting enumeration space

- Strengthened Herbrand Theorem allows restriction to $\mathcal{T}(E)$

- \textit{Sort inference} reduces instantiation space by computing more precise sort information
  - $E \cup Q = \{a \neq b, f(a) = c\} \cup \{P(f(x))\}$
    - $a, b, c, x : \tau$
    - $f : \tau \rightarrow \tau$ and $P : \tau \rightarrow \text{Bool}$
  - This is equivalent to $E^s \cup Q^s = \{a_1 \neq b_1, f_{12}(a_1) = c_2\} \cup \{P_2(f_{12}(x_1))\}$
    - $a_1, b_1, x_1 : \tau_1$
    - $c_2 : \tau_2$
    - $f_{12} : \tau_1 \rightarrow \tau_2$ and $P : \tau_2 \rightarrow \text{Bool}$

- \textbf{u} would derive e.g. $x \mapsto c$ for $E \cup Q$, while for $E^s \cup Q^s$ the instantiation $x_1 \mapsto c_2$ is not well-sorted
Two-layered method for checking whether $E \models \varphi[\bar{x}/\bar{t}]$ holds

- cache of instances already derived
- on-the-fly rewriting of $\varphi[\bar{x}/\bar{t}]$ modulo $E$
  with extension to other theories through theory-specific rewriting
Advanced $u$: term ordering

Instances are enumerated according to the order

$$(t_1, \ldots, t_n) \prec (s_1, \ldots, s_n) \text{ if } \begin{cases} \max_{i=1}^{n} t_i \prec \max_{i=1}^{n} s_i, \text{ or} \\ \max_{i=1}^{n} t_i = \max_{i=1}^{n} s_i \text{ and} \\
(t_1, \ldots, t_n) \prec_{\text{lex}} (s_1, \ldots, s_n) \end{cases}$$

for a given order $\preceq$ on ground terms.

If $a \prec b \prec c$, then

$$(a, a) \prec (a, b) \prec (b, a) \prec (b, b) \prec (a, c) \prec (c, b) \prec (c, c)$$

- instances with $c$ considered only after considering all cases with $a$ and $b$
- goal is to introduce new terms less often
- order on $T(E)$ fixed for finite set of terms $t_1 < \ldots < t_n$
  - instantiate in order with $t_1, \ldots, t_n$
  - then choose new non-congruent term $t \in T(E)$ and have $t_n \prec t$
- Still a lot of room for improvement (and ML?) [Janota21]
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
- E-matching/trigger-based instantiation (e)
- Conflict-based instantiation (c)
- Model-based instantiation (m)
- Enumerative instantiation (u)

Experimental evaluation

Conclusion

References
Experimental evaluation (UNSAT)

CVC4 configurations on unsatisfiable benchmarks

- 42,065 benchmarks: 14,731 TPTP + 27,334 SMT-LIB
- e+u: interleave e and u
- e;u: apply e first, then u if it fails
- All CVC4 configurations have c; as prefix
Experimental evaluation (SAT)

<table>
<thead>
<tr>
<th>Library</th>
<th>#</th>
<th>u</th>
<th>e;u</th>
<th>e+u</th>
<th>e</th>
<th>m</th>
<th>e;m</th>
<th>e+m</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPTP</td>
<td>14731</td>
<td>471</td>
<td>492</td>
<td>464</td>
<td>17</td>
<td>930</td>
<td>808</td>
<td>829</td>
</tr>
<tr>
<td>UF</td>
<td>7293</td>
<td>39</td>
<td>42</td>
<td>42</td>
<td>0</td>
<td>70</td>
<td>69</td>
<td>65</td>
</tr>
<tr>
<td>Theories</td>
<td>20041</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>350</td>
<td>267</td>
<td>267</td>
</tr>
<tr>
<td>Total</td>
<td>42065</td>
<td>513</td>
<td>537</td>
<td>509</td>
<td>20</td>
<td>1350</td>
<td>1144</td>
<td>1161</td>
</tr>
</tbody>
</table>
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
   E-matching/trigger-based instantiation (e)
   Conflict-based instantiation (c)
   Model-based instantiation (m)
   Enumerative instantiation (u)
   Experimental evaluation

Conclusion

References
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
   E-matching/trigger-based instantiation (e)
   Conflict-based instantiation (c)
   Model-based instantiation (m)
   Enumerative instantiation (u)
   Experimental evaluation

Conclusion

References
Conclusion

- Quantifiers in SMT: handled in an ad hoc manner
- Techniques presented here are pure FOL with equality (i.e. not “Modulo Theories”)
- Reasonably effective nonetheless

Future works and perspectives

- New instantiation techniques (Vampire-like attitude, in SMT?)
- Machine learning
- More convergence with state-of-the-art FOL techniques from saturation theorem proving
- Symbiosis with quantifier elimination for theory reasoning
- Convergence with FOL provers?
- Higher-order logic
COST EU Action EuroProofNet

https://europroofnet.github.io/

EuroProofNet aims at boosting the interoperability and usability of proof systems
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
   E-matching/trigger-based instantiation \( (e) \)
   Conflict-based instantiation \( (c) \)
   Model-based instantiation \( (m) \)
   Enumerative instantiation \( (u) \)
   Experimental evaluation

Conclusion

References
Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques
   E-matching/trigger-based instantiation (e)
   Conflict-based instantiation (c)
   Model-based instantiation (m)
   Enumerative instantiation (u)

Experimental evaluation

Conclusion

References
References


References II


References III


