An Overview of Quantifier Instantiation in Modern SMT Solvers

Andrew Reynolds
May 28, 2021
Satisfiability Modulo Theories (SMT) Solvers

- SMT solvers are:
  - Fully automated reasoners
  - Widely used in applications

Software Verification Tools
- Verification Conditions

Interactive Proof Assistants
- Conjectures

Symbolic Execution Engines
- Path Constraints

Synthesis Tools, Planners
- Specifications
SMT Solvers

• Traditionally:
  • Efficient decision procedures for *quantifier-free* constraints over theories:
    • Arithmetic
    • Uninterpreted functions (UF)
    • Bitvectors
    • Arrays
    • Datatypes
    • More recently: strings, floating points, sets, relations, ...

• In the past decade:
  • Efficient (heuristic) techniques for *quantified* formulas as well

⇒ Focus of this talk
Applications of ∀ in SMT

• Are used for:
  • Automated theorem proving:
    • Background axioms \{∀x.g(e,x)=g(x,e)=x, ∀x.g(x,g(y,z))=g(g(x,y),x), ∀x.g(x,i(x))=e\}
  • Software verification:
    • Unfolding ∀x.foo(x)=bar(x+1), code contracts ∀x.pre(x)⇒post(f(x))
    • Frame axioms ∀x.x≠t ⇒ A’(x)=A(x)
  • Function Synthesis:
    • Synthesis conjectures ∀i:input.∃o:output.R[o,i]
  • Planning:
    • Specifications ∃p:plan.∀t:time.F[P,t]
SMT Solvers for $\forall$ using Quantifier Instantiation

• Traditionally:

• More recently:
  • Model-Based Instantiation [Ge et al 2009, Reynolds et al 2013]
  • Conflict-Based Instantiation [Reynolds et al 2014, Barbosa et al 2017]
  • Theory-specific Approaches
    • Linear arithmetic [Bjorner 2012, Reynolds et al 2015, Janota et al 2015]
    • Bit-Vectors [Wintersteiger et al 2013, Dutertre 2015]

Implemented in
simplify, z3, FX7, Alt-Ergo, Princess, cvc5, veriT, SMTInterpol
SMT Solvers for $\forall$ using Quantifier Instantiation

• Traditionally:

• More recently:
  • Conflict-Based Instantiation [Reynolds et al 2014, Barbosa et al 2017]
  • Model-Based Instantiation [Ge et al 2009, Reynolds et al 2013]
  • Enumerative Instantiation [Reynolds et al 2018]
  • Counterexample-Guided / QE [Reynolds et al 2015, Janota et al 2015]
  • Syntax-Guided [Preiner et al 2017, Niemetz et al 2021]

Implemented in
  • simplify, z3, FX7, Alt-Ergo, Princess, cvc5, veriT, SMTInterpol
  • cvc5, veriT, SMTInterpol
  • z3, cvc5
  • cvc5, veriT
  • z3, cvc5, yices
  • boolector, cvc5
DPLL(T)-Based SMT Solvers (quantifier-free)
DPLL(T)-Based SMT Solvers

T-Clauses $F$

Context $M$

QF Solver

SAT Solver

Theory solver(s)

...when $F$ is unsatisfiable

$M \models p F$
DPLL(T)-Based SMT Solvers

T-Clauses $F, F_1 \ldots F_n$

Context $M$

QF Solver

SAT Solver

Theory solver(s)

conflicts, lemmas

...when $M$ is T-satisfiable

sat
DPLL(T)-Based SMT Solvers

T-Clauses $F \ F_1 \ldots \ F_n$

Context $\mathcal{M}$

QF Solver

SAT Solver

Theory solver(s)

...when $\mathcal{M}$ is $T$-satisfiable

...when $F$ is unsatisfiable

unsat

sat
DPLL(T)-Based SMT Solvers + $\forall$ Instantiation

$T$-Clauses $F$

QF Solver

SAT Solver $\rightarrow$ Theory solver(s)

Context $M$

...when $F$ is unsatisfiable

...when $M$ is $T$-satisfiable
DPLL(T)-Based SMT Solvers + ∀ Instantiation

T-Clauses $F$

QF Solver

SAT Solver

Theory solver(s)

Context $M$

When $M$ contains quantified formulas $\forall$...

...when $F$ is unsatisfiable

...when $M$ is $T$-satisfiable
DPLL(T)-Based SMT Solvers + $\forall$ Instantiation

**T-Clauses $F$**

- **QF Solver**
  - **SAT Solver**
  - **Theory solver(s)**

**Context $M$**

- **unsat**
  - ...when $F$ is unsatisfiable

- **sat?**
  - ...when $M$ is T-satisfiable

**Undecidability!**

- ...cannot always establish $M$ is sat
DPLL(T)-Based SMT Solvers + \( \forall \) Instantiation

QF Solver

T-Clauses \( F \)

SAT Solver → Theory solver(s) → Context \( M \)

...when \( F \) is unsatisfiable
DPLL(T)-Based SMT Solvers + $\forall$ Instantiation

Set of ground formulas
- $\{ f(a)=b, P(a), \ldots \}$

Set of quantified formulas
- $\{ \forall x . P(x), \ldots \}$

unsat

...when $F$ is unsatisfiable
DPLL(T)-Based SMT Solvers + $\forall$ Instantiation

T-Clauses $F$

QF Solver

SAT Solver

Theory solver(s)

Context $M$

$E$

$Q$

$\forall$ Solver

unsat

...when $F$ is unsatisfiable
DPLL(T)-Based SMT Solvers + ∀ Instantiation

T-Clauses $F$

QF Solver

SAT Solver
Theory solver(s)

Context $M$

∀ Solver

$F_1 \ldots F_n$

Instantiation lemmas

unsat

...when $F$ is unsatisfiable

Given $\forall x. P(x)$ in $Q$, instantiation lemma $F_i$ is a valid formula:

$\forall x. P(x) \Rightarrow P(t)$

for some ground term $t$
DPLL(T)-Based SMT Solvers + $\forall$ Instantiation

T-Clauses $F$

QF Solver

SAT Solver

Theory solver(s)

Context $M$

$\forall$ Solver

$E$

$Q$

...when $F$ is unsatisfiable

...when $E, Q$ is T-satisfiable

(Instantiation) lemmas

$F_1, ..., F_n$
DPLL(T)-Based SMT Solvers + ∀ Instantiation

- T-Clauses $F$
- QF Solver
  - SAT Solver
  - Theory solver(s)
- Context $\mathcal{M}$
- Solver
  - $E$
  - $Q$
  - $F_1 \ldots F_n$
- unsat
  - When can we answer “unsat”? • Which lemmas are likely lead to “unsat”?
- sat
  - ...when $E, Q$ is $T$-satisfiable
Techniques for Quantifier Instantiation

∀ Solver

- Conflict-Based
- E-matching
- Model-Based
- Enumerative

Instances of ∀ in Q

E ⊕ Q is T-satisfiable

sat
∀ Solver

- Conflict-Based
- E-matching
- Model-Based
- Enumerative
- CEX-Guided
- Syntax-Guided
E-matching

- **Idea:** Instantiations found by pattern matching $\mathcal{Q}$ to terms from $\mathcal{E}$

- Implemented in early SMT solvers (e.g. simplify) as well as z3, cvc5

- **Key applications:** Software verification
  - Example: Dafny/Boogie
E-matching

\[ E \quad \begin{cases} P(a) \\ \neg P(b) \\ P(c) \end{cases} \quad Q \quad \forall x. P(x) \lor R(x) \]

E-matching
E-matching

\[ \forall x. \neg P(x) \Rightarrow \neg P(a) \]
\[ \forall x. \neg P(x) \Rightarrow \neg P(b) \]
\[ \forall x. \neg P(x) \Rightarrow \neg P(c) \]
E-matching: Impact

• Highly effective for quantifiers with UF
  • Widely used as backend for many software verification applications

• Challenges:
  • Pattern selection, multi-patterns
  • *Too many* instances produced, non-termination (matching loops)
    • ...solver times out
  • *Incomplete*
    • ...solver answers “unknown”
Conflict-Based Instantiation

• **Idea:** Find instantiation that is in conflict with $E$, if it exists

• A *conflicting instance* forces the solver to backtrack
  • Improves ability to answer “unsat”

• Implemented in cvc5, veriT
  • [Reynolds et al FMCAD 2014, Barbosa et al TACAS 2017]

• **Key applications:** Automated Theorem Proving
  • Example: Isabelle/Sledgehammer
Conflict-Based Instantiation

\[ E \rightarrow \neg P(b) \quad \neg P(a) \rightarrow \neg P(c) \quad \forall x. P(x) \rightarrow \text{Conflict-Based} \]
Conflict-Based Instantiation

\[ \exists x. P(x) \iff P(a) \land \neg P(b) \land P(c) \]

\[ \forall x. P(x) \Rightarrow P(a) \]
\[ \forall x. P(x) \Rightarrow P(b) \]
\[ \forall x. P(x) \Rightarrow P(c) \]

If no conflicting instance exists, resort to E-matching

\[ \neg P(b), P(b) \models \bot \]

Conflicting instance
Conflict-Based Instantiation: Impact

- Using conflict-based instantiation (cvc4+ci), require an order of magnitude fewer instances for showing “UNSAT” wrt E-matching alone

(taken from [Reynolds et al FMCAD14], evaluation On SMTLIB, TPTP, Isabelle benchmarks)
Conflict-Based Instantiation: Impact

- CVC4 with conflicting instances **cvc4+ci**
  - Solves the **most benchmarks** for TPTP and Isabelle
  - Requires almost an order of magnitude **fewer instantiations**

<table>
<thead>
<tr>
<th></th>
<th>TPTP</th>
<th>Isabelle</th>
<th>SMT-LIB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solved</td>
<td>Inst</td>
<td>Solved</td>
</tr>
<tr>
<td><strong>cvc3</strong></td>
<td>5,245</td>
<td>627.0M</td>
<td>3,827</td>
</tr>
<tr>
<td><strong>z3</strong></td>
<td>6,269</td>
<td>613.5M</td>
<td>3,506</td>
</tr>
<tr>
<td><strong>cvc4</strong></td>
<td>6,100</td>
<td>879.0M</td>
<td>3,858</td>
</tr>
<tr>
<td><strong>cvc4+ci</strong></td>
<td><strong>6,616</strong></td>
<td><strong>150.9M</strong></td>
<td><strong>4,082</strong></td>
</tr>
</tbody>
</table>

⇒ A number of hard benchmarks can be solved without resorting to E-matching at all
∀ Solver

Conflict-Based
E-matching
Model-Based
Enumerative

CEX-Guided
Syntax-Guided
Model-Based Instantiation

- **Idea:** Instantiate quantifiers based on (complete) models for $E$

- Complete for certain fragments, e.g. EPR, essentially uninterpreted
  - Can be useful for answering “sat”

- Implemented in z3, finite model finding in cvc4
  - [Ge et al 2009, Reynolds et al 2013]

- **Key applications:** Software Design, Planning
  - Example: Alloy Analyzer
Model-Based Instantiation

$\neg R(a)$
$\neg R(b)$

$\forall x. P(x)$
Model-Based Instantiation

\( E \)
- \( \neg R(a) \)
- \( \neg R(b) \)

\( Q \)
- \( \forall x. P(x) \)

\( R = \lambda x. \text{false} \)
\( P = \lambda x. \text{true} \)

\[ \text{Model-Based} \]

\[ \Rightarrow \text{Resort to model-based only when E-matching saturates} \]
Model-based Instantiation: Impact

- CVC4 Finite Model Finding + Model-Based instantiation [Reynolds et al CADE13]
  - Approach can scale to domains of >2 billion, only adds fraction of possible instances
∀ Solver

- Conflict-Based
- E-matching
- Model-Based
- Enumerative

- CEX-Guided
- Syntax-Guided
Enumerative Instantiation

• **Idea:** Instantiate based on (fair) enumeration of terms from $E$

• Effective alternative to model-based, better performance for “unsat”

• Complete for limited fragments

• Implemented in cvc5, veriT
  • [Reynolds et al TACAS 2018, Janota et al 2021]

• Key applications: Automated theorem proving
  • Example: Isabelle/Sledgehammer, TPTP
Enumerative Instantiation

\[ f(a) = b \]
\[ P(h(b)) \]

\[ \forall x. P(f(x)) \]
Enumerative Instantiation

\(\forall x. P(f(x)) \Rightarrow P(f(a))\)
\(\forall x. P(f(x)) \Rightarrow P(f(b))\)
\(\forall x. P(f(x)) \Rightarrow P(f(f(a)))\)
\(\forall x. P(f(x)) \Rightarrow P(f(h(b)))\)

\(\Rightarrow\) Finds instances that E-matching may miss, more lightweight than MBQI

Ordering over terms from \(E\):

\(a < b < f(a) < h(b) < \ldots\)
Solver

- Conflict-Based
- E-matching
  - Model-Based
  - Enumerative

Generally, used for $\forall$ with UF logics

- CEX-Guided
- Syntax-Guided

Generally, used for $\forall$ in pure theories
Counterexample-Guided Instantiation

• **Idea:** Instantiate based on T-solving for counterexamples $\neg Q \land E$

• Can be seen as a lazy quantifier elimination algorithm in SMT loop

• Complete for quantified linear integer/real arithmetic, finite domains

• Variants of idea implemented in cvc5, (extensions of) z3, yices

• Key applications: Synthesis, Hardware Verification, Compiler Optimization
Counterexample-guided Instantiation

\[ \forall x. x+b > a \]
Counterexample-guided Instantiation

\[ E \quad \begin{cases} \quad a > b \\ \quad x + b \leq a \end{cases} \]

\[ Q \quad \forall x . x + b > a \]

Solve for \( x \)

\[ x = a - b \]

\[ \forall x . x + b > a \Rightarrow (a - b) + b > a \]

where:

\[ (a - b) + b > a \iff a > a \iff \bot \]

\[ \Rightarrow \text{Can simulate e.g. Cooper, Loos-Weispfenning, Ferrante-Rackoff algorithms for QE} \]
Syntax-Guided Instantiation

• **Idea:** Instantiate based on enumerating terms from T-specific grammar

• Leverages advances in syntax-guided synthesis (SyGuS) [Alur et al 2013]

• **Implemented:**
  • For bitvector theory in Boolector [Preiner et al TACAS 2017]
  • For all supported theories in cvc5 [Niemetz et al TACAS 2021]

• **Key applications:** Synthesis and Verification for emerging theories
  • E.g. quantifiers over floating points
Syntax-Guided Instantiation

\[ \forall x. x^2 \neq a^2 + b^2 + 2ab \]

none
Syntax-Guided Instantiation

$$\forall x. x^2 \neq a^2 + b^2 + 2*a*b$$

Construct grammar generating terms of integer type

$$G \rightarrow a \mid b \mid 0 \mid 1 \mid G+G \mid G*G \mid -G$$

Best known approach for theories where QE is unknown
Summary

• SMT solvers handle diverse set of inputs (with quantifiers)

• Best instantiation technique depends on the logic
  • When UF is present:
    ⇒ *E*-matching, conflict-based, model-based, enumerative
  • For traditional theories (e.g. LIA, BV) which emit quantifier elimination:
    ⇒ Counterexample-guided
  • For other theories (e.g. floating points, strings, non-linear arithmetic):
    ⇒ Syntax-guided
• Techniques in this talk implemented in SMT solver cvc5
  • Open source
  • Available at: https://github.com/cvc5

• ...Thanks for listening!