New techniques for instantiation and proof production in SMT solving

Haniel Barbosa

University of Lorraine, CNRS, Inria, LORIA, Nancy, France
PPgSC, DIMAp, UFRN, Natal, Brazil
Advisors: Pascal Fontaine, David Déharbe, and Stephan Merz

PhD defense
2017–09–05, Nancy, France
Automated Reasoning

- Formal Verification
- Program Analysis
- Automatic Testing
- Program Synthesis
Formal Verification

Program Analysis

Automatic Testing

Program Synthesis

SMT Solvers
Problem statement

\[ \varphi = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor (q(a) \land \neg q(b + x))] \]
Problem statement

ϕ = a ≤ b ∧ b ≤ a + x ∧ x ≃ 0 ∧ [f(a) \not\equiv f(b) ∨ (q(a) ∧ ¬q(b + x))]  

Clausified formula:

ϕ′ = a ≤ b ∧ b ≤ a + x ∧ x ≃ 0 ∧ [f(a) \not\equiv f(b) ∨ q(a)] ∧ [f(a) \not\equiv f(b) ∨ ¬q(b + x)]
Problem statement

\[ \varphi = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor (q(a) \land \neg q(b + x))] \]

Clausified formula:

\[ \varphi' = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor q(a)] \land [f(a) \not\simeq f(b) \lor \neg q(b + x)] \]

Propositional abstraction:

\[ \text{abs}(\varphi') = p_{a \leq b} \land p_{b \leq a + x} \land p_{x \simeq 0} \land (\neg p_{f(a) \simeq f(b)} \lor p_{q(a)}) \land (\neg p_{f(a) \simeq f(b)} \lor \neg p_{q(b + x)}) \]
Problem statement

\[ \varphi = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor (q(a) \land \neg q(b + x))] \]

Clausified formula:

\[ \varphi' = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor q(a)] \land [f(a) \not\simeq f(b) \lor \neg q(b + x)] \]

Propositional abstraction:

\[ \operatorname{abs}(\varphi') = p_{a \leq b} \land p_{b \leq a + x} \land p_{x \simeq 0} \land (\neg p_{f(a) \simeq f(b)} \lor p_{q(a)}) \land (\neg p_{f(a) \simeq f(b)} \lor \neg p_{q(b + x)}) \]

Satisfying assignment:

\[ \{ p_{a \leq b}, p_{b \leq a + x}, p_{x \simeq 0}, \neg p_{f(a) \simeq f(b)} \} \Rightarrow \{ a \leq b, b \leq a + x, x \simeq 0, f(a) \not\simeq f(b) \} \]
Problem statement

\( \varphi = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor (q(a) \land \neg q(b + x))] \)

Clausified formula:

\( \varphi' = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor q(a)] \land [f(a) \not\simeq f(b) \lor \neg q(b + x)] \)

Propositional abstraction:

\[ abs(\varphi') = \neg p_{a \leq b} \land \neg p_{b \leq a + x} \land \neg p_{x \simeq 0} \land (\neg f(a) \simeq f(b)) \lor p_{q(a)} \land (\neg f(a) \simeq f(b)) \lor \neg p_{q(b + x)} \]

Satisfying assignment:

\[ \{ p_{a \leq b}, p_{b \leq a + x}, p_{x \simeq 0}, \neg p_{f(a) \simeq f(b)} \} \Rightarrow \{ a \leq b, b \leq a + x, x \simeq 0, f(a) \not\simeq f(b) \} \]

Conflict clause:

\[ \neg (a \leq b) \lor \neg (b \leq a + x) \lor \neg (x \simeq 0) \lor f(a) \simeq f(b) \]
Problem statement

\[ \varphi = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor (q(a) \land \neg q(b + x))] \]

Clausified formula:

\[ \varphi' = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor q(a)] \land [f(a) \not\simeq f(b) \lor \neg q(b + x)] \]

\[ \varphi'' = \varphi' \land \neg(a \leq b) \lor \neg(b \leq a + x) \lor \neg(x \simeq 0) \lor f(a) \simeq f(b) \]
Problem statement

\[ \varphi = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor (q(a) \land \neg q(b + x))] \]

Clausified formula:

\[ \varphi' = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor q(a)] \land [f(a) \not\simeq f(b) \lor \neg q(b + x)] \]

\[ \varphi'' = \varphi' \land \neg (a \leq b) \lor \neg (b \leq a + x) \lor \neg (x \simeq 0) \lor f(a) \simeq f(b) \]

Satisfying assignment: \{a \leq b, b \leq a + x, x \simeq 0, q(a), \neg q(b + x)\}
Problem statement

\[ \varphi = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor (q(a) \land \neg q(b + x))] \]

Clausified formula:

\[ \varphi' = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\simeq f(b) \lor q(a)] \land [f(a) \not\simeq f(b) \lor \neg q(b + x)] \]

\[ \varphi'' = \varphi' \land \neg(a \leq b) \lor \neg(b \leq a + x) \lor \neg(x \simeq 0) \lor f(a) \simeq f(b) \]

Satisfying assignment: \{a \leq b, b \leq a + x, x \simeq 0, q(a), \neg q(b + x)\}

Conflict clause: \neg(a \leq b) \lor \neg(b \leq a + x) \lor \neg(x \simeq 0) \lor \neg q(a) \lor q(b + x)
Problem statement

\[ \varphi = a \leq b \land b \leq a + x \land x \approx 0 \land [f(a) \not\approx f(b) \lor (q(a) \land \neg q(b + x))] \]

Clausified formula:

\[ \varphi' = a \leq b \land b \leq a + x \land x \approx 0 \land [f(a) \not\approx f(b) \land q(a)] \land [f(a) \not\approx f(b) \lor \neg q(b + x)] \]

\[ \varphi'' = \varphi' \land \neg (a \leq b) \lor \neg (b \leq a + x) \lor \neg (x \approx 0) \lor f(a) \approx f(b) \]

\[ \varphi'' = \varphi'' \land \neg (a \leq b) \lor \neg (b \leq a + x) \lor \neg (x \approx 0) \lor \neg q(a) \lor q(b + x) \]
Problem statement

$$\varphi = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\equiv f(b) \lor (q(a) \land \neg q(b + x))]$$

Clausified formula:

$$\varphi' = a \leq b \land b \leq a + x \land x \simeq 0 \land [f(a) \not\equiv f(b) \lor q(a)] \land [f(a) \not\equiv f(b) \lor \neg q(b + x)]$$

$$\varphi'' = \varphi' \land \neg(a \leq b) \lor \neg(b \leq a + x) \lor \neg(x \simeq 0) \lor f(a) \simeq f(b)$$

$$\varphi''' = \varphi'' \land \neg(a \leq b) \lor \neg(b \leq a + x) \lor \neg(x \simeq 0) \lor \neg q(a) \lor q(b + x)$$

UNSAT!
Quantifier-free solver enumerates models $E$

- $E$ is a set of ground literals

\[
\{ a \leq b, \ b \leq a + x, \ x \approx 0, \ f(a) \not\approx f(b) \}\]
Problem statement

Quantifier-free solver enumerates models $E \cup Q$

- $E$ is a set of ground literals
  \begin{equation*}
  \{a \leq b, \ b \leq a + x, \ x \simeq 0, \ f(a) \not\simeq f(b)\}
  \end{equation*}

- $Q$ is a set of quantified clauses
  \begin{equation*}
  \{\forall xyz. \ f(x) \not\simeq f(z) \lor g(y) \simeq h(z)\}
  \end{equation*}

Instantiation module generates instances of $Q$

\begin{equation*}
  f(a) \not\simeq f(b) \lor g(a) \simeq h(b)
  \end{equation*}
A unifying framework for instantiating quantified formulas with equality and uninterpreted functions

[B., Fontaine, Reynolds. TACAS’17]

(I1) Formalizing underlying problem for instantiation in SMT

(I2) Lifting congruence closure to accommodate free variables

(I3) Casting existing instantiation techniques in framework

(I4) Techniques for efficient implementation
Contributions

Scalable fine-grained proofs for formula processing

[B., Blanchette, Fontaine. CADE’17]

(P1) Extensible inference system for formula processing

(P2) Proof producing generic contextual recursion algorithm

(P3) Proving desirable properties of rules and algorithms

(P4) Validation of framework through implementation and prototype checker
Contribution 1: A unifying framework for instantiating quantified formulas with equality and uninterpreted functions
Pattern-matching of terms from $Q$ into terms of $E$

for $\forall xyz. f(x) \not\equiv f(z) \lor g(y) \simeq h(z)$ a pattern is \{f(x), g(y), h(z)\}

- Fast, but too many instances

$E$ with $10^2$ applications each for $f$, $g$, $h$ leads to up to $10^6$ instantiations
Heuristic instantiation

Pattern-matching of terms from $Q$ into terms of $E$

for $\forall xyz. f(x) \not\approx f(z) \lor g(y) \simeq h(z)$ a pattern is \{f(x), g(y), h(z)\}

- Fast, but too many instances

\(E\) with \(10^2\) applications each for \(f, g, h\) leads to up to \(10^6\) instantiations
Heuristic instantiation

Pattern-matching of terms from $Q$ into terms of $E$

$$\forall xyz. f(x) \not\approx f(z) \lor g(y) \simeq h(z)$$

a pattern is $\{f(x), g(y), h(z)\}$

- Fast, but too many instances

  $E$ with $10^2$ applications each for $f$, $g$, $h$ leads to up to $10^6$ instantiations
Pattern-matching of terms from $Q$ into terms of $E$

for $\forall xyz. f(x) \not\approx f(z) \lor g(y) \simeq h(z)$ a pattern is \{f(x), g(y), h(z)\}

- Fast, but too many instances

$E$ with $10^2$ applications each for $f$, $g$, $h$ leads to up to $10^6$ instantiations
Heuristic instantiation

Pattern-matching of terms from $Q$ into terms of $E$

for $\forall xyz. f(x) \not\approx f(z) \lor g(y) \simeq h(z)$ a pattern is $\{f(x), g(y), h(z)\}$

- Fast, but too many instances
  
  $E$ with $10^2$ applications each for $f$, $g$, $h$ leads to up to $10^6$ instantiations
Heuristic instantiation

Pattern-matching of terms from $Q$ into terms of $E$

for $\forall xyz. f(x) \not\equiv f(z) \lor g(y) \simeq h(z)$ a pattern is $\{f(x), g(y), h(z)\}$

- Fast, but too many instances

$E$ with $10^2$ applications each for $f$, $g$, $h$ leads to up to $10^6$ instantiations
Heuristic instantiation

Pattern-matching of terms from $Q$ into terms of $E$

for $\forall xyz. f(x) \not\equiv f(z) \lor g(y) \simeq h(z)$ a pattern is \{f(x), g(y), h(z)\}

- Fast, but too many instances

$E$ with $10^2$ applications each for $f, g, h$ leads to up to $10^6$ instantiations

Easily gets out of hand!
Goal-oriented instantiation

Check consistency of $E \cup Q$

- Only instances refuting the current model are generated
Goal-oriented instantiation

Check consistency of $E \cup Q$

- Only instances refuting the current model are generated
  
  If $\{f(a) \simeq f(c), g(b) \not\simeq h(c)\} \subseteq E$, then $E$ is refuted with the instantiation
  
  $\forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \rightarrow f(a) \not\simeq f(c) \lor g(b) \simeq h(c)$
Goal-oriented instantiation

Check consistency of $E \cup Q$

- Only instances refuting the current model are generated

If \( \{f(a) \simeq f(c), g(b) \not\simeq h(c)\} \subseteq E \), then $E$ is refuted with the instantiation

\[
\forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \rightarrow f(a) \not\simeq f(c) \lor g(b) \simeq h(c)
\]
Goal-oriented instantiation

Check consistency of $E \cup Q$

- Only instances refuting the current model are generated
  
  If $\{f(a) \simeq f(c), g(b) \not\simeq h(c)\} \subseteq E$, then $E$ is refuted with the instantiation
  
  $\forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \rightarrow f(a) \not\simeq f(c) \lor g(b) \simeq h(c)$
Goal-oriented instantiation

Check consistency of $E \cup Q$

- Only instances refuting the current model are generated
  - If $\{f(a) \simeq f(c), g(b) \not\simeq h(c)\} \subseteq E$, then $E$ is refuted with the instantiation
    $\forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \rightarrow f(a) \not\simeq f(c) \lor g(b) \simeq h(c)$

\[ \forall \bar{x}. \psi \rightarrow \psi\sigma \]

\[ E \land \psi\sigma \models \bot \]

Goal-oriented instantiation module
Goal-oriented instantiation

Check consistency of $E \cup Q$

- Only instances refuting the current model are generated

If $\{f(a) \simeq f(c), g(b) \not\simeq h(c)\} \subseteq E$, then $E$ is refuted with the instantiation

$$\forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \rightarrow f(a) \not\simeq f(c) \lor g(b) \simeq h(c)$$

$$\forall x. \psi \rightarrow \psi \sigma$$

$$E \land \psi \sigma \models \bot$$

UNSAT!
Previous work

Conflict-based instantiation [RTM14]

▷ Given a model $E \cup Q$, for some $\forall \bar{x}. \psi \in Q$ find $\sigma$ s.t. $E \land \psi\sigma \models \bot$

▷ Add instance $\forall \bar{x}. \psi \rightarrow \psi\sigma$ to quantifier-free solver

Finding conflicting instances requires deriving $\sigma$ s.t.

$$E \models \neg\psi\sigma$$

⊕ Goal-oriented

⊕ Efficient

⊖ Ad-hoc

⊖ Incomplete
Let’s look deeper into the problem

\[ E \models \neg \psi \sigma, \text{ for some } \forall \bar{x}. \psi \in Q \]
Let’s look deeper into the problem

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

\[ E = \{ f(a) \simeq f(c), g(b) \not\simeq h(c) \}, \quad Q = \{ \forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \} \]
Let’s look deeper into the problem

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

\[ E = \{f(a) \simeq f(c), g(b) \not\simeq h(c)\}, \quad Q = \{\forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z)\} \]

\[ f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma \]
Let’s look deeper into the problem

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

\[ E = \{f(a) \simeq f(c), g(b) \not\simeq h(c)\}, \quad \mathcal{Q} = \{\forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z)\} \]

\[ f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma \]

\[ \triangleright \text{ Each literal in the right hand side delimits possible } \sigma \]
Let’s look deeper into the problem

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

\[ E = \{ f(a) \simeq f(c), g(b) \not\simeq h(c) \}, \quad Q = \{ \forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \} \]

\[ f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma \]

▷ Each literal in the right hand side delimits possible \( \sigma \)

▷ \( f(x) \simeq f(z) \): either \( x \simeq z \) or \( x \simeq a \land z \simeq c \) or \( x \simeq c \land z \simeq a \)
Let's look deeper into the problem

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

\[ E = \{ f(a) \simeq f(c), g(b) \not\simeq h(c) \}, \quad Q = \{ \forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \} \]

\[ f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma \]

▶ Each literal in the right hand side delimits possible \( \sigma \)

▶ \( f(x) \simeq f(z) \): either \( x \simeq z \) or \( x \simeq a \land z \simeq c \) or \( x \simeq c \land z \simeq a \)

▶ \( g(y) \not\simeq h(z) \): \( y \simeq b \land z \simeq c \)
Let’s look deeper into the problem

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

\[ E = \{ f(a) \simeq f(c), \ g(b) \not\simeq h(c) \} \]
\[ \mathcal{Q} = \{ \forall xyz. \ f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \} \]

\[ f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma \]

▷ Each literal in the right hand side delimits possible \( \sigma \)

▷ \( f(x) \simeq f(z) \): either \( x \simeq z \) or \( x \simeq a \land z \simeq c \) or \( x \simeq c \land z \simeq a \)

▷ \( g(y) \not\simeq h(z) \): \( y \simeq b \land z \simeq c \)

\[ \sigma = \{ x \mapsto c, \ y \mapsto b, \ z \mapsto c \} \]
Let’s look deeper into the problem

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

\[ E = \{ f(a) \simeq f(c), g(b) \not\simeq h(c) \}, \quad Q = \{ \forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \} \]

\[ f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma \]

\( \triangleright \) Each literal in the right hand side delimits possible \( \sigma \)

\( \triangleright \) \( f(x) \simeq f(z) \): either \( x \simeq z \) or \( x \simeq a \land z \simeq c \) or \( x \simeq c \land z \simeq a \)

\( \triangleright \) \( g(y) \not\simeq h(z) \): \( y \simeq b \land z \simeq c \)

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

or

\[ \sigma = \{ x \mapsto a, y \mapsto b, z \mapsto c \} \]
Let’s look deeper into the problem

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

\[ E = \{ f(a) \simeq f(c), g(b) \not\simeq h(c) \}, \ Q = \{ \forall xyz. f(x) \not\simeq f(z) \lor g(y) \simeq h(z) \} \]

\[ f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma \]

▷ Each literal in the right hand side delimits possible \( \sigma \)

▷ \( f(x) \simeq f(z) \): either \( x \simeq z \) or \( x \simeq a \land z \simeq c \) or \( x \simeq c \land z \simeq a \)

▷ \( g(y) \not\simeq h(z) \): \( y \simeq b \land z \simeq c \)

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

or

\[ \sigma = \{ x \mapsto a, y \mapsto b, z \mapsto c \} \]
Given conjunctive sets of equality literals $E$ and $L$, with $E$ ground, finding a substitution $\sigma$ s.t. $E \models L\sigma$
Given conjunctive sets of equality literals $E$ and $L$, with $E$ ground, finding a substitution $\sigma$ s.t. $E \models L\sigma$

▷ Solution space can be restricted into ground terms from $E \cup L$
Given conjunctive sets of equality literals $E$ and $L$, with $E$ ground, finding a substitution $\sigma$ s.t. $E \models L\sigma$

- Solution space can be restricted into ground terms from $E \cup L$

- NP-complete

  - NP: solutions can checked in polynomial time
  - NP-hard: reduction of 3-SAT into the entailment
Given conjunctive sets of equality literals $E$ and $L$, with $E$ ground, finding a substitution $\sigma$ s.t. $E \models L\sigma$

- Solution space can be restricted into ground terms from $E \cup L$

- NP-complete
  
  NP: solutions can checked in polynomial time
  NP-hard: reduction of 3-SAT into the entailment

- Variant of classic (non-simultaneous) rigid $E$-unification

$$s_1\sigma \simeq t_1\sigma, \ldots, s_n\sigma \simeq t_n\sigma \models u\sigma \simeq v\sigma$$
Congruence Closure with Free Variables (CCFV) is a sound, complete and terminating calculus for solving $E$-ground (dis)unification.
Congruence Closure with Free Variables (CCFV) is a sound, complete and terminating calculus for solving $E$-ground (dis)unification

- Goal-oriented
- Efficient
Congruence Closure with Free Variables (CCFV) is a sound, complete and terminating calculus for solving $E$-ground (dis)unification.

- **Goal-oriented**
- **Efficient**
- **Ad-hoc**: Versatile framework, recasting many instantiation techniques as a CCFV problem
- **Incomplete**: Finds all conflicting instances of a quantified formula
Existing techniques as special cases

- **Conflict-based instantiation** [RTM14]
  - CCFV provides formal guarantees and more clear extensions

- **$E$-matching based heuristic instantiation** [DNS05; MB07]
  - CCFV allows to easily discard instances already entailed by $E$

- **Model-based instantiation** [GM09; RTG+13]
  - No need for a secondary ground SMT solver
  - No need to guess solutions
Finding solutions $\sigma$ for $E \models L\sigma$
Finding solutions $\sigma$ for $E \models L\sigma$

$$f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma$$

$$f(x) \simeq f(z) \land g(y) \not\simeq h(z)$$
Finding solutions $\sigma$ for $E \models L\sigma$

\[
\begin{align*}
E &\models L\sigma \\
 f(a) \simeq f(c) \land g(b) \not\simeq h(c) &\models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma \\
 f(x) \simeq f(z) \land g(y) \not\simeq h(z) &\models \varnothing \\
 f(x) \simeq f(z) \land z \simeq c \land y \simeq b
\end{align*}
\]
Finding solutions $\sigma$ for $E \models L\sigma$

\[ f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma \]

\[ f(x) \simeq f(z) \land g(y) \not\simeq h(z) \]

\[ \emptyset \]

\[ f(x) \simeq f(z) \land z \simeq c \land y \simeq b \]

\[ y \simeq b \]

\[ f(x) \simeq f(z) \land z \simeq c \]
Finding solutions $\sigma$ for $E \models L\sigma$

\[
E \models L\sigma
\]

\[
f(a) \simeq f(c) \land g(b) \not\simeq h(c) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma
\]

\[
f(x) \simeq f(z) \land g(y) \not\simeq h(z)
\]

\[
\varnothing
\]

\[
f(x) \simeq f(z) \land z \simeq c \land y \simeq b
\]

\[
y \simeq b
\]

\[
f(x) \simeq f(z) \land z \simeq c
\]

\[
y \simeq b, \ z \simeq c
\]

\[
f(x) \simeq f(c)
\]
Finding solutions $\sigma$ for $E \models L\sigma$

\[
\begin{align*}
f(a) & \simeq f(c) \land g(b) \not\simeq h(c) \\
(f(x) & \simeq f(z) \land g(y) \not\simeq h(z)) \sigma
\end{align*}
\]

\[
\begin{align*}
f(x) & \simeq f(z) \land g(y) \not\simeq h(z) \\
\emptyset & \\
f(x) & \simeq f(z) \land z \simeq c \land y \simeq b \\
y & \simeq b \\
f(x) & \simeq f(z) \land z \simeq c \\
y & \simeq b, z \simeq c \\
f(x) & \simeq f(c)
\end{align*}
\]

$x \simeq a$  $x \simeq c$
Finding solutions $\sigma$ for $E \models L\sigma$

\[
\begin{align*}
E & \models L\sigma \\
\rightarrow & \quad (f(a) \simeq f(c) \land g(b) \not\simeq h(c)) \models (f(x) \simeq f(z) \land g(y) \not\simeq h(z)) \sigma
\end{align*}
\]

\[
\begin{align*}
\rightarrow & \quad f(x) \simeq f(z) \land g(y) \not\simeq h(z) \\
\rightarrow & \quad \varnothing \\
\rightarrow & \quad f(x) \simeq f(z) \land z \simeq c \land y \simeq b \\
\rightarrow & \quad y \simeq b \\
\rightarrow & \quad f(x) \simeq f(z) \land z \simeq c \\
\rightarrow & \quad y \simeq b, z \simeq c \\
\rightarrow & \quad f(x) \simeq f(c)
\end{align*}
\]

$x \simeq a$

$x \simeq c$

$x \simeq a, y \simeq b, z \simeq c$

$\top$

$x \simeq c, y \simeq b, z \simeq c$

$\top$
Model minimisation
Model minimisation

Top symbol indexing of $E$-graph from ground congruence closure

\[ E \models f(x)\sigma \sim t \text{ only if } [t] \text{ contains some } f(t') \]

\[ f \rightarrow \left\{ \begin{array}{l}
    f([t_1], \ldots, [t_n]) \\
    \vdots \\
    f([t'_1], \ldots, [t'_n])
\end{array} \right. \]

Bitsets for fast checking if a symbol has applications in a congruence class
Selection strategies

$E \models f(x, y) \simeq h(z) \land x \simeq t \land \ldots$
Selection strategies

\[ E \models f(x, y) \simeq h(z) \land x \simeq t \land \ldots \]

Eagerly checking whether constraints can be discarded

- After assigning \( x \) to \( t \), the remaining problem is normalized

\[ E \models f(t, y) \simeq h(z) \land \ldots \]

- \( E \models f(t, y) \sigma \simeq h(z) \sigma \) only if there is some \( f(t', t'') \) s.t.

\[ E \models t \simeq t' \]
Implementation

A breadth-first implementation of CCFV:

- Explores sets of solutions at a time

\[ E \models \ell_1 \land \ldots \land \ell_n \]

\[ \downarrow \quad \downarrow \]

\[ \mathcal{S}_1 \quad \square \quad \ldots \quad \square \quad \mathcal{S}_n \quad \text{individual solutions for each literal} \]

\[ \mathcal{G} \quad \text{combination of compatible solutions} \]

- Heavy use of memoization
- Bottleneck in merging solution sets
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.
The depth-first CCFV outperforms its breadth-first counterpart by a small margin.

Both perform well and are viable approaches

* experiments in the “UF”, “UFLIA”, “UFLRA” and “UFIDL” categories of SMT-LIB, which have 10 495 benchmarks annotated as unsatisfiable, with 100s timeout.
Contributions
A unifying framework for quantified formulas with equality and uninterpreted functions

Formalizing underlying problem for instantiation in SMT
Lifting congruence closure to accommodate free variables
Casting existing instantiation techniques in framework
Efficient implementations in the SMT solvers veriT and CVC4
Contributions

A unifying framework for quantified formulas with equality and uninterpreted functions

- Formalizing underlying problem for instantiation in SMT
- Lifting congruence closure to accommodate free variables
- Casting existing instantiation techniques in framework
- Efficient implementations in the SMT solvers veriT and CVC4

Extensions

- Incrementality
- Learning-based search for solutions
- Finding conflicting instances across multiple quantified formulas

\[ E \models \neg \psi_1 \sigma \lor \cdots \lor \neg \psi_n \sigma, \quad \forall \bar{x}. \psi \in Q \]

- Beyond theory of equality
- Handle variables in \( E \)
Contribution 2: Scalable fine-grained proofs for formula processing
Why proofs?

- to check the result for unsatisfiable/valid formulas
- for solver/prover cooperation

- as a debugging facility
- for evaluation purposes (how good is the algorithm?)

- as a part of the reasoning framework (e.g. conflict clauses)
- to extract cores
- to compute interpolants
Challenges for proofs in FOL

▷ Collecting and storing proof information efficiently

▷ Producing proofs for sophisticated processing techniques

▷ Producing proofs for modules that use external tools

▷ Standardizing a proof format
Challenges for proofs in FOL

- Collecting and storing proof information efficiently
  no convergence, but quite active
  [KBT+16; HBR+15; MB08; BODF09; SZS04; Sch13; KV13; WDF+09]

- Producing proofs for sophisticated processing techniques
  proofs with holes or too coarse

- Producing proofs for modules that use external tools
  depends on tool

- Standardizing a proof format
  open
Challenges for proofs in FOL

▷ Collecting and storing proof information efficiently
  no convergence, but quite active
  [KBT+16; HBR+15; MB08; BODF09; SZS04; Sch13; KV13; WDF+09]

▷ Producing proofs for sophisticated processing techniques
  proofs with holes or too coarse  scalable fine-grained proofs

▷ Producing proofs for modules that use external tools
  depends on tool

▷ Standardizing a proof format
  open
Proofs in veriT

Resolution chains, input formulas, tautologies for theory and quantifier reasoning
Proofs in veriT

Resolution chains, input formulas, tautologies for theory and quantifier reasoning

▷ SAT solver: resolution

\[
\begin{array}{c}
A \lor \ell \\
B \lor \bar{\ell}
\end{array}
\quad \frac{}{A \lor B}
\]

Antecedents: \( A \lor \ell, \ B \lor \bar{\ell} \)

Pivot: \( \ell \) or \( \bar{\ell} \)

Resolvent: \( A \lor B = (A \lor \ell) \diamond (B \lor \bar{\ell}) \)
Proofs in veriT

Resolution chains, input formulas, tautologies for theory and quantifier reasoning

▷ SAT solver: resolution

\[
\frac{A \lor \ell \quad B \lor \overline{\ell}}{A \lor B}
\]

Antecedents: \( A \lor \ell, B \lor \overline{\ell} \)

Pivot: \( \ell \) or \( \overline{\ell} \)

Resolvent: \( A \lor B = (A \lor \ell) \diamond (B \lor \overline{\ell}) \)

▷ theory solvers: theory lemmas

\( \neg(a \simeq c) \lor \neg(c \simeq b) \lor a \simeq b \quad \neg(a \simeq b) \lor f(a) \simeq f(b) \)

\( \neg(y > 1) \lor \neg(x < 1) \lor y > x \)
Proofs in veriT

Resolution chains, input formulas, tautologies for theory and quantifier reasoning

▷ SAT solver: resolution

\[
\begin{array}{c}
A \lor \ell \\
B \lor \overline{\ell}
\end{array}
\overline{\ell} \quad \begin{array}{c}
A \lor \ell \\
B \lor \overline{\ell}
\end{array}
\overline{\ell}
\Rightarrow
\begin{array}{c}
A \lor B
\end{array}
\]

Antecedents: \( A \lor \ell, B \lor \overline{\ell} \)

Pivot: \( \ell \) or \( \overline{\ell} \)

Resolvent: \( A \lor B = (A \lor \ell) \diamond (B \lor \overline{\ell}) \)

▷ theory solvers: theory lemmas

\[ \neg(a \simeq c) \lor \neg(c \simeq b) \lor a \simeq b \]
\[ \neg(a \simeq b) \lor f(a) \simeq f(b) \]
\[ \neg(y > 1) \lor \neg(x < 1) \lor y > x \]

▷ instantiation module: instantiation lemmas

\[ \neg(\forall x. \psi[x]) \lor \psi[t] \]
Proving formula processing

- Resolution does not capture all transformations
- Some transformations do not preserve logical equivalence
- Code is lengthy and deals with many cases
- Difficult to manipulate binders soundly and efficiently
Proving formula processing

Contributions
Scalable fine-grained proofs for formula processing

- Resolution does not capture all transformations
- Some transformations do not preserve logical equivalence
- Code is lengthy and deals with many cases
- Difficult to manipulate binders soundly and efficiently

Extensible framework to produce proofs for processing techniques involving *locally replacing equals by equals* in the presence of *binders*

Some instances:

- Skolemization: \((\neg \forall x. p(x)) \simeq \neg p(\varepsilon x. \neg p(x))\)

- let elimination: \((\text{let } x \simeq a \text{ in } p(x, x)) \simeq p(a, a)\)

- theory simplifications: \((k + 1 \times 0 < k) \simeq (k < k)\)
A context $\Gamma$ fixes a set of variables and specifies a substitution

\[
\Gamma ::= \emptyset \mid \Gamma, x \mid \Gamma, x_n \mapsto s_n
\]
A context $\Gamma$ fixes a set of variables and specifies a substitution

$$\Gamma ::= \emptyset \mid \Gamma, x \mid \Gamma, \bar{x}_n \mapsto \bar{s}_n$$

Rules have the form

$$\begin{array}{c}
\frac{D_1 \ldots D_n}{\Gamma \vdash t \simeq u} R
\end{array}$$

半导体ly, the judgement expresses the equality of the terms $\Gamma(t)$ and $u$ for all variables fixed by $\Gamma$
Example of ‘let’ expansion

\[ \triangleright a \simeq a \]

**Cong**

\[ \frac{x \mapsto a \triangleright x \simeq a \quad \text{Refl} \\ x \mapsto a \triangleright p(x, x) \simeq p(a, a)} {\triangleright \left( \text{let } x \simeq a \text{ in } p(x, x) \right) \simeq p(a, a)} \]

**Refl**

\[ \frac{x \mapsto a \triangleright x \simeq a \quad \text{Cong}} {x \mapsto a \triangleright x \simeq a} \]

\[ \frac{x \mapsto a \triangleright p(x, x) \simeq p(a, a)} {\triangleright \left( \text{let } x \simeq a \text{ in } p(x, x) \right) \simeq p(a, a)} \]

**LET**
Example of theory simplification

\[
\begin{align*}
\triangleright k & \simeq k \quad \text{CONG} \\
\triangleright 1 \times 0 & \simeq 0 \quad \text{TAUT}_\times \\
\triangleright k + 1 \times 0 & \simeq k + 0 \quad \text{CONG} \\
\triangleright k + 0 & \simeq k \quad \text{TAUT}_+ \\
\triangleright k + 1 \times 0 & \simeq k \quad \text{TRANS} \\
\triangleright (k + 1 \times 0 \leq k) & \simeq (k < k) \quad \text{CONG}
\end{align*}
\]
Example of skolemization

The skolemization proof of the formula \( \neg \forall x. p(x) \):

\[
\begin{array}{c}
\frac{x \mapsto \varepsilon x. \neg p(x) \triangleright x \simeq \varepsilon x. \neg p(x)}{x \mapsto \varepsilon x. \neg p(x) \triangleright p(x) \simeq p(\varepsilon x. \neg p(x))} \\
\triangleright (\forall x. p(x)) \simeq p(\varepsilon x. \neg p(x)) \\
\triangleright (\neg \forall x. p(x)) \simeq \neg p(\varepsilon x. \neg p(x))
\end{array}
\]

veriT syntax:

\[
(.c0 (Sko_All :conclusion ((\forall x. p(x)) \simeq p(\varepsilon x. \neg p(x))))
 :args (x \mapsto (\varepsilon x. \neg p(x))))
 :subproof ((.c1 (Refl :conclusion (x \simeq (\varepsilon x. \neg p(x))))))
 (\cdot.c2 (Cong :clauses (.c1)
 :conclusion (p(x) \simeq p(\varepsilon x. \neg p(x))))))
 (\cdot.c3 (Cong :clauses (.c0) :conclusion ((\neg \forall x. p(x)) \simeq \neg p(\varepsilon x. \neg p(x))))))
\]
function \textit{process}(\Delta, t) \\
match t \\
\hspace{1em} \text{case } x: \hspace{1em} \text{return } \textit{build\_var}(\Delta, x) \\
\hspace{1em} \text{case } f(\bar{t}_n): \hspace{1em} \bar{\Delta}'_n \leftarrow (\textit{ctx\_app}(\Delta, f, \bar{t}_n, i))_{i=1}^n \\
\hspace{1em} \text{return } \textit{build\_app}(\Delta, \bar{\Delta}'_n, f, \bar{t}_n, (\textit{process}(\Delta'_i, t_i))_{i=1}^n) \\
\hspace{1em} \text{case } Qx.\varphi: \hspace{1em} \Delta' \leftarrow \textit{ctx\_quant}(\Delta, Q, x, \varphi) \\
\hspace{1em} \text{return } \textit{build\_quant}(\Delta, \Delta', Q, x, \varphi, \textit{process}(\Delta', \varphi)) \\
\hspace{1em} \text{case } \text{let } \bar{x}_n \simeq \bar{r}_n \text{ in } t': \hspace{1em} \Delta' \leftarrow \textit{ctx\_let}(\Delta, \bar{x}_n, \bar{r}_n, t') \\
\hspace{1em} \text{return } \textit{build\_let}(\Delta, \Delta', \bar{x}_n, \bar{r}_n, t', \textit{process}(\Delta', t'))
Soundness of inference rules proven through an encoding into simply typed $\lambda$-calculus

$$M ::= t \mid \lambda x. M \mid (\lambda \bar{x}. M) \bar{t}_n$$

$$\frac{D_1 \quad \cdots \quad D_n}{M \simeq N} \quad \text{R}$$
Soundness of inference rules proven through an encoding into simply typed $\lambda$-calculus

\[ M ::= t \mid \lambda x. M \mid (\lambda \overline{x}_n. M) \overline{t}_n \]

\[ \frac{D_1 \quad \cdots \quad D_n}{M \simeq N} \]

Correctness of proof-producing contextual recursion algorithm

Cost of proof production is linear and of proof checking is (almost) linear*

* assuming suitable data structures
Proof output for veriT

Framework implemented with a proof-producing contextual recursion algorithm

- fine-grained proofs for most processing transformations
- No negative impact on performance
- More transformations in proof producing mode
- Dramatic simplification of the code base

Prototype checker in Isabelle/HOL

Maps proofs into Isabelle theorems

- Judgements encoded in $\lambda$-calculus
Summary

- Centralizes manipulation of bound variables and substitutions
- Accommodates many transformations (e.g. Skolemization)
- Proof checking is (almost) linear
- Implementation and integration within veriT

Contributions

Scalable fine-grained proofs for formula processing

[CADE'17]
Summary

 Contributions
 [CADE’17]
 Scalable fine-grained proofs for formula processing

▷ Centralizes manipulation of bound variables and substitutions
▷ Accommodates many transformations (e.g. Skolemization)
▷ Proof checking is (almost) linear
▷ Implementation and integration within veriT

Future work

▷ Support global rewritings within the framework
▷ Support richer logics (e.g. HOL)
▷ Implement proof reconstruction in Isabelle/HOL
Conclusion

- Extensible framework for handling instantiation in SMT solving
- Extensible framework for proving formula processing in SMT solving
- Successful implementations
- Publications at TACAS’17 and CADE’17, pending submission to JAR
References


