New techniques for instantiation and proof production in SMT solving

Haniel Barbosa

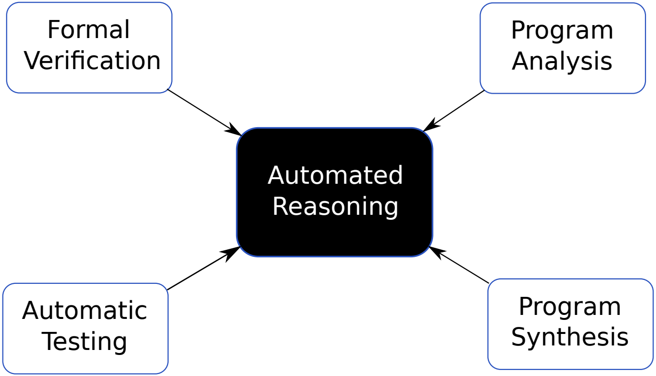
University of Lorraine, CNRS, Inria, LORIA, Nancy, France

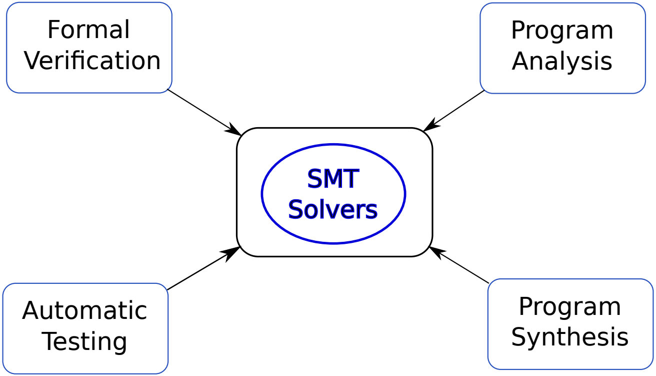
PPgSC, DIMAp, UFRN, Natal, Brazil

Advisors: Pascal Fontaine, David Déharbe, and Stephan Merz

PhD defense

2017–09–05, Nancy, France


Formal Verification
Program Analysis
Automated Reasoning
Automatic Testing
Program Synthesis


Formal Verification
Program Analysis
SMT Solvers
Automatic Testing
Program Synthesis

Problem statement

$$\varphi = a \leq b \wedge b \leq a + x \wedge x \simeq 0 \wedge [f(a) \not\simeq f(b) \vee (q(a) \wedge \neg q(b + x))]$$

Problem statement

$$\varphi = a \leq b \wedge b \leq a + x \wedge x \simeq 0 \wedge [f(a) \not\simeq f(b) \vee (q(a) \wedge \neg q(b + x))]$$

Clausified formula:

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

Problem statement

$$\varphi = a \leq b \wedge b \leq a + x \wedge x \simeq 0 \wedge [f(a) \not\simeq f(b) \vee (q(a) \wedge \neg q(b + x))]$$

Clausified formula:

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

Propositional abstraction:

$$abs(\varphi') = p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x \simeq 0} \wedge (\neg p_{f(a) \simeq f(b)} \vee p_{q(a)}) \wedge (\neg p_{f(a) \simeq f(b)} \vee \neg p_{q(b+x)})$$

Problem statement

$$\varphi = a \leq b \wedge b \leq a + x \wedge x \simeq 0 \wedge [f(a) \not\simeq f(b) \vee (q(a) \wedge \neg q(b + x))]$$

Clausified formula:

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

Propositional abstraction:

$$abs(\varphi') = p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x \simeq 0} \wedge (\neg p_{f(a) \simeq f(b)} \vee p_{q(a)}) \wedge (\neg p_{f(a) \simeq f(b)} \vee \neg p_{q(b+x)})$$

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

$$\varphi = a \leq b \wedge b \leq a + x \wedge x \simeq 0 \wedge [f(a) \not\simeq f(b) \vee (q(a) \wedge \neg q(b + x))]$$

Clausified formula:

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

Propositional abstraction:

$$abs(\varphi') = p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x \simeq 0} \wedge (\neg p_{f(a) \simeq f(b)} \vee p_{q(a)}) \wedge (\neg p_{f(a) \simeq f(b)} \vee \neg p_{q(b+x)})$$

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: $\neg(a \leq b) \vee \neg(b \leq a + x) \vee \neg(x \simeq 0) \vee f(a) \simeq f(b)$

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

$$\varphi = a \leq b \wedge b \leq a + x \wedge x \simeq 0 \wedge [f(a) \not\simeq f(b) \vee (q(a) \wedge \neg q(b + x))]$$

Clausified formula:

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

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

Problem statement

$\varphi = a \leq b \wedge b \leq a + x \wedge x \simeq 0 \wedge [f(a) \not\simeq f(b) \vee (q(a) \wedge \neg q(b + x))]$

Clausified formula:

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

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

Conflict clause: $\neg(a \leq b) \vee \neg(b \leq a + x) \vee \neg(x \simeq 0) \vee \neg q(a) \vee q(b + x)$

Problem statement

$$\varphi = a \leq b \wedge b \leq a + x \wedge x \simeq 0 \wedge [f(a) \not\simeq f(b) \vee (q(a) \wedge \neg q(b + x))]$$

Clausified formula:

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

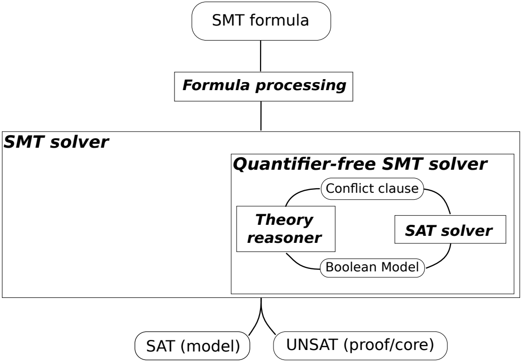
Problem statement

$$\varphi = a \leq b \wedge b \leq a + x \wedge x \simeq 0 \wedge [f(a) \not\simeq f(b) \vee (q(a) \wedge \neg q(b+x))]$$

Clausified formula:

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

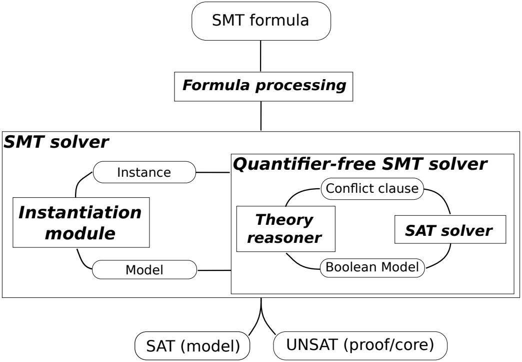
Problem statement



Quantifier-free solver enumerates models E

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

Problem statement



Quantifier-free solver enumerates models $E \cup \mathcal{Q}$

- E is a set of ground literals $\{a \leq b,\, b \leq a + x,\, x \simeq 0,\, f(a) \not\simeq f(b)\}$
- $\mathcal{Q}$ is a set of quantified clauses $\{\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z)\}$

Instantiation module generates instances of $\mathcal{Q}$ $f(a) \not\simeq f(b) \vee g(a) \simeq h(b)$

Contributions

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]

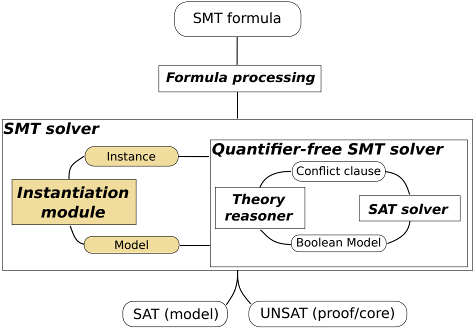
(P1) Extensible inference system for formula processing

(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



Heuristic instantiation

Pattern-matching of terms from $\mathcal{Q}$ into terms of E

for $\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ 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

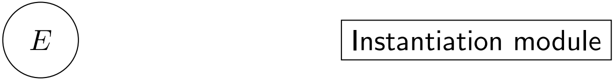
Heuristic instantiation

Pattern-matching of terms from $\mathcal{Q}$ into terms of E

for $\forall xyz.\ f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ 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



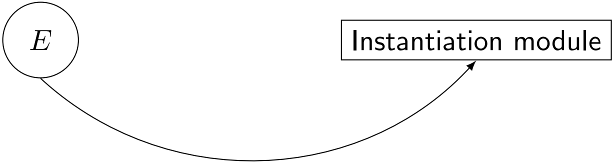
Heuristic instantiation

Pattern-matching of terms from $\mathcal{Q}$ into terms of E

for $\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ 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



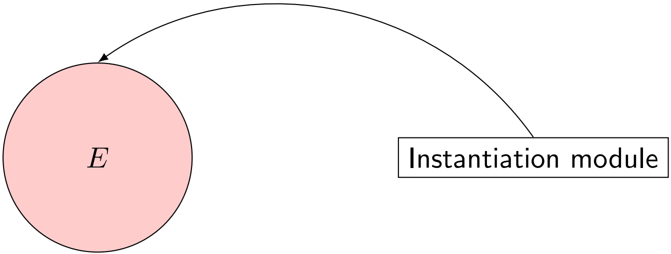
Heuristic instantiation

Pattern-matching of terms from $\mathcal{Q}$ into terms of E

for $\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ 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



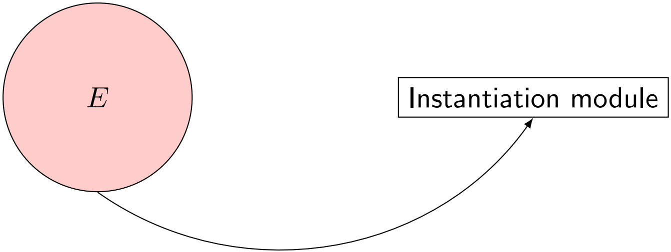
Heuristic instantiation

Pattern-matching of terms from $\mathcal{Q}$ into terms of E

for $\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ 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



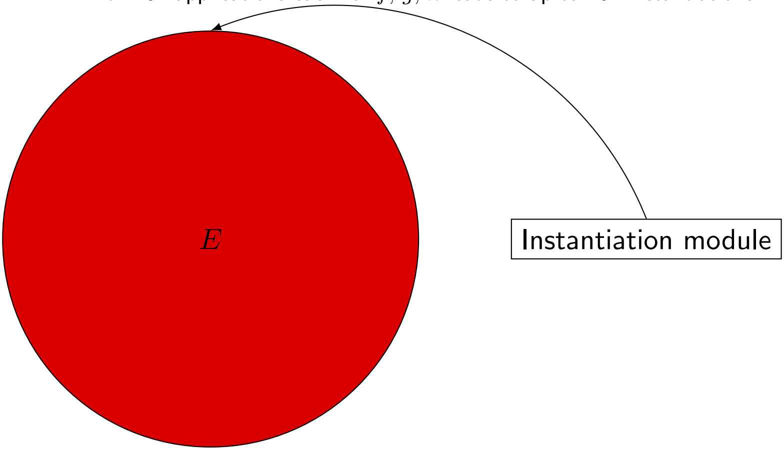
Heuristic instantiation

Pattern-matching of terms from $\mathcal{Q}$ into terms of E

for $\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ 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



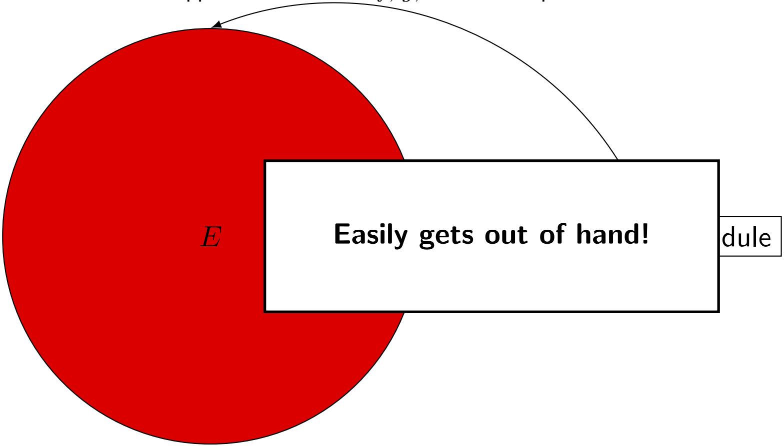
Heuristic instantiation

Pattern-matching of terms from $\mathcal{Q}$ into terms of E

for $\forall xyz.\ f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ 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 $E \cup \mathcal{Q}$

- $\oplus$ Only instances refuting the current model are generated

Goal-oriented instantiation

Check consistency of $E \cup \mathcal{Q}$

- $\oplus$ 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

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

Goal-oriented instantiation

Check consistency of $E \cup \mathcal{Q}$

- $\oplus$ 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

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

E

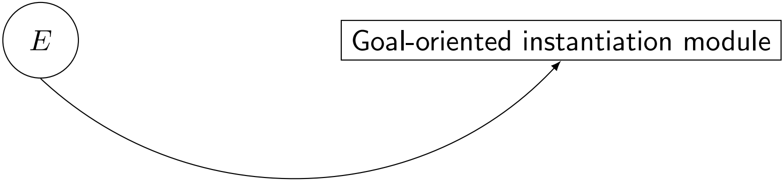
Goal-oriented instantiation module

Goal-oriented instantiation

Check consistency of $E \cup \mathcal{Q}$

- $\oplus$ 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
  $\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z) \to f(a) \not\simeq f(c) \vee g(b) \simeq h(c)$



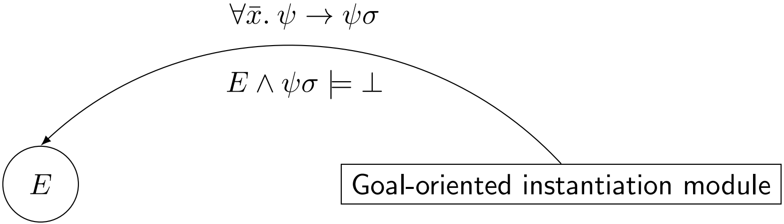
Goal-oriented instantiation

Check consistency of $E \cup \mathcal{Q}$

- $\oplus$ 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

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



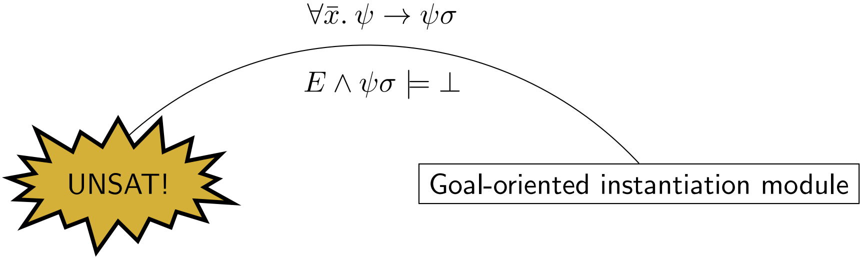
Goal-oriented instantiation

Check consistency of $E \cup \mathcal{Q}$

- $\oplus$ 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

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



Previous work

Conflict-based instantiation [RTM14]

- Given a model $E \cup \mathcal{Q}$, for some $\forall \bar{x}.\ \psi \in \mathcal{Q}$ find σ s.t. $E \wedge \psi\sigma \models \bot$
- Add instance $\forall \bar{x}.\ \psi \rightarrow \psi\sigma$ to quantifier-free solver

Finding conflicting instances requires deriving σ s.t.

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

- $\oplus$ Goal-oriented
- $\oplus$ Efficient
- $\ominus$ Ad-hoc
- $\ominus$ Incomplete

Let's look deeper into the problem

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

Let's look deeper into the problem

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

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

Let's look deeper into the problem

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

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

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

Let's look deeper into the problem

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

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

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

- ▷ Each literal in the right hand side delimits possible σ

Let's look deeper into the problem

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

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

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

- ▷ Each literal in the right hand side delimits possible σ
  - ▶ $f(x) \simeq f(z)$: either $x \simeq z$ or $x \simeq a \wedge z \simeq c$ or $x \simeq c \wedge z \simeq a$

Let's look deeper into the problem

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

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

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

- Each literal in the right hand side delimits possible σ
  - $f(x) \simeq f(z)$: either $x \simeq z$ or $x \simeq a \wedge z \simeq c$ or $x \simeq c \wedge z \simeq a$
  - $g(y) \not\simeq h(z)$: $y \simeq b \wedge z \simeq c$

Let's look deeper into the problem

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

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

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

- Each literal in the right hand side delimits possible σ
  - $f(x) \simeq f(z)$: either $\underline{x \simeq z}$ or $x \simeq a \wedge z \simeq c$ or $x \simeq c \wedge z \simeq a$
  - $g(y) \not\simeq h(z)$: $\underline{y \simeq b \wedge z \simeq c}$

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

Let's look deeper into the problem

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

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

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

- ▷ Each literal in the right hand side delimits possible σ
  - ▶ $f(x) \simeq f(z)$: either $x \simeq z$ or $\underline{x \simeq a \wedge z \simeq c}$ or $x \simeq c \wedge z \simeq a$
  - ▶ $g(y) \not\simeq h(z)$: $\underline{y \simeq b \wedge z \simeq c}$

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

or

$$\sigma = \{x \mapsto a,\ y \mapsto b,\ z \mapsto c\}$$

Let's look deeper into the problem

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

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

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

- Each literal in the right hand side delimits possible σ
  - $f(x) \simeq f(z)$: either $x \simeq z$ or $x \simeq a \wedge z \simeq c$ or $\underline{x \simeq c \wedge z \simeq a}$
  - $g(y) \not\simeq h(z)$: $\underline{y \simeq b \wedge z \simeq c}$

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

or

$$\sigma = \{x \mapsto a,\ y \mapsto b,\ z \mapsto c\}$$

E-ground (dis)unification

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

E-ground (dis)unification

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

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

E-ground (dis)unification

Given conjunctive sets of equality literals E and L, with E ground, finding a substitution σ 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

E-ground (dis)unification

Given conjunctive sets of equality literals E and L, with E ground, finding a substitution σ 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

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

Congruence Closure with Free Variables

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

- $\oplus$ Goal-oriented
- $\oplus$ Efficient

Congruence Closure with Free Variables

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

- $\oplus$ Goal-oriented
- $\oplus$ Efficient
- $\ominus$ ~~Ad-hoc~~ **Versatile framework, recasting many instantiation techniques as a CCFV problem**
- $\ominus$ ~~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 σ for $E \models L\sigma$

$$
\begin{array}{rcl}
E & \models & L\sigma \\
f(a) \simeq f(c) \wedge g(b) \not\simeq h(c) & \models & (f(x) \simeq f(z) \wedge g(y) \not\simeq h(z))\, \sigma
\end{array}
$$

Finding solutions σ for $E \models L\sigma$

$$\begin{array}{rcl} E & \models & L\sigma \\ f(a) \simeq f(c) \wedge g(b) \not\simeq h(c) & \models & (f(x) \simeq f(z) \wedge g(y) \not\simeq h(z))\,\sigma \end{array}$$

$$f(x) \simeq f(z) \wedge g(y) \not\simeq h(z)$$

Finding solutions σ for $E \models L\sigma$

$$
\begin{array}{rcl}
E & \models & L\sigma \\
f(a) \simeq f(c) \wedge g(b) \not\simeq h(c) & \models & (f(x) \simeq f(z) \wedge g(y) \not\simeq h(z))\,\sigma
\end{array}
$$

$$
\begin{array}{c}
f(x) \simeq f(z) \wedge g(y) \not\simeq h(z) \\
\emptyset \Big| \\
f(x) \simeq f(z) \wedge z \simeq c \wedge y \simeq b
\end{array}
$$

Finding solutions σ for $E \models L\sigma$

$$\begin{array}{rcl} E & \models & L\sigma \\ f(a) \simeq f(c) \wedge g(b) \not\simeq h(c) & \models & (f(x) \simeq f(z) \wedge g(y) \not\simeq h(z))\,\sigma \end{array}$$

$$f(x) \simeq f(z) \wedge g(y) \not\simeq h(z)$$

$$\emptyset \;\Big|$$

$$f(x) \simeq f(z) \wedge z \simeq c \wedge y \simeq b$$

$$y \simeq b \;\Big|$$

$$f(x) \simeq f(z) \wedge z \simeq c$$

Finding solutions σ for $E \models L\sigma$

$$\begin{array}{rcl} E & \models & L\sigma \\ f(a) \simeq f(c) \wedge g(b) \not\simeq h(c) & \models & (f(x) \simeq f(z) \wedge g(y) \not\simeq h(z))\,\sigma \end{array}$$

$$f(x) \simeq f(z) \wedge g(y) \not\simeq h(z)$$

$$\varnothing$$

$$f(x) \simeq f(z) \wedge z \simeq c \wedge y \simeq b$$

$$y \simeq b$$

$$f(x) \simeq f(z) \wedge z \simeq c$$

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

$$f(x) \simeq f(c)$$

Finding solutions σ for $E \models L\sigma$

$$
\begin{array}{rcl}
E & \models & L\sigma \\
f(a) \simeq f(c) \wedge g(b) \not\simeq h(c) & \models & (f(x) \simeq f(z) \wedge g(y) \not\simeq h(z))\,\sigma
\end{array}
$$

$$f(x) \simeq f(z) \wedge g(y) \not\simeq h(z)$$

$$\varnothing$$

$$f(x) \simeq f(z) \wedge z \simeq c \wedge y \simeq b$$

$$y \simeq b$$

$$f(x) \simeq f(z) \wedge z \simeq c$$

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

$$f(x) \simeq f(c)$$

$$x \simeq a \qquad\qquad x \simeq c$$

Finding solutions σ for $E \models L\sigma$

$$
\begin{array}{rcl}
E & \models & L\sigma \\
f(a) \simeq f(c) \wedge g(b) \not\simeq h(c) & \models & (f(x) \simeq f(z) \wedge g(y) \not\simeq h(z))\,\sigma
\end{array}
$$

$$f(x) \simeq f(z) \wedge g(y) \not\simeq h(z)$$

$$\Big\downarrow{\scriptstyle \varnothing}$$

$$f(x) \simeq f(z) \wedge z \simeq c \wedge y \simeq b$$

$$\Big\downarrow{\scriptstyle y \simeq b}$$

$$f(x) \simeq f(z) \wedge z \simeq c$$

$$\Big\downarrow{\scriptstyle y \simeq b,\ z \simeq c}$$

$$f(x) \simeq f(c)$$

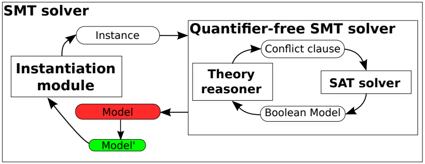
$$x \simeq a \qquad\qquad x \simeq c$$

$$x \simeq a,\ y \simeq b,\ z \simeq c \Big\downarrow \qquad\qquad \Big\downarrow x \simeq c,\ y \simeq b,\ z \simeq c$$

$$\top \qquad\qquad \top$$

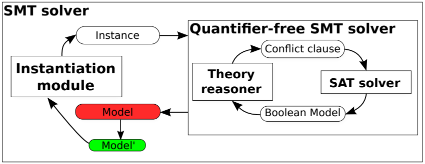
Implementation

▷ Model minimisation



Implementation

▷ Model minimisation



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

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

$$f \to \begin{cases} f([t_1], \ldots, [t_n]) \\ \ldots \\ f([t'_1], \ldots, [t'_n]) \end{cases}$$

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

Implementation

- ▷ Selection strategies

$$E \models f(x, y) \simeq h(z) \wedge x \simeq t \wedge \ldots$$

Implementation

- Selection strategies

$$E \models f(x, y) \simeq h(z) \wedge x \simeq t \wedge \ldots$$

- Eagerly checking whether constraints can be discarded
  - After assigning x to t, the remaining problem is normalized

$$E \models f(t, y) \simeq h(z) \wedge \ldots$$

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

$$E \models t \simeq t'$$

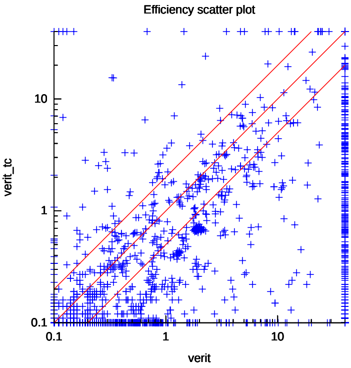
A breadth-first implementation of CCFV:

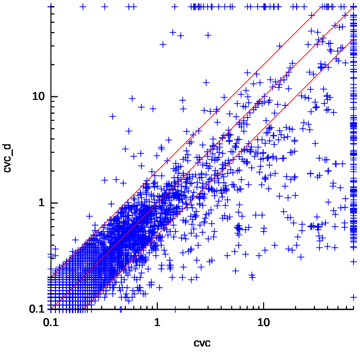
▷ Explores sets of solutions at a time

$$
\begin{array}{ccccccccc}
E & \models & \ell_1 & \wedge & \ldots & \wedge & \ell_n & & \\
 & & \downarrow & & & & \downarrow & & \\
 & & \multicolumn{5}{c}{\underbrace{\mathfrak{S}_1 \quad \sqcap \quad \ldots \quad \sqcap \quad \mathfrak{S}_n}_{\mathfrak{S}}} & & \text{individual solutions for each literal} \\
 & & & & & & & & \text{combination of compatible solutions}
\end{array}
$$

⊕ Heavy use of memoization

⊖ Bottleneck in merging solution sets

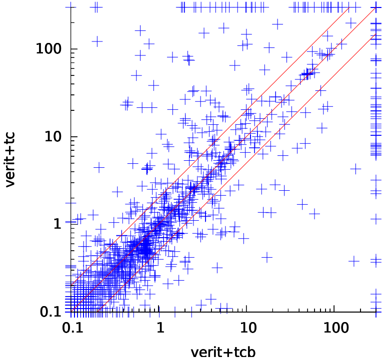




veriT: + 800 out of $1\,785$ unsolved problems

CVC4:+ 200 out of 745 unsolved problems

* experiments in the "UF", "UFLIA", "UFLRA" and "UFIDL" categories of SMT-LIB, which have $10\,495$ 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.

Summary

- ▷ 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

Summary

- ▷ 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

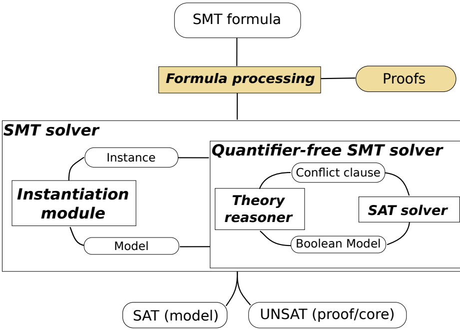
Extensions

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

$$E \models \neg\psi_1\sigma \vee \cdots \vee \neg\psi_n\sigma, \quad \forall\bar{x}.\ \psi \in \mathcal{Q}$$

- ▷ 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

$$\frac{A \vee \ell \qquad B \vee \bar{\ell}}{A \vee B}$$

Antecedents: $A \vee \ell$, $B \vee \bar{\ell}$
Pivot: ℓ or $\bar{\ell}$
Resolvent: $A \vee B = (A \vee \ell) \diamond (B \vee \bar{\ell})$

Proofs in veriT

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

- SAT solver: resolution

$$\frac{A \vee \ell \qquad B \vee \overline{\ell}}{A \vee B}$$

Antecedents: $A \vee \ell$, $B \vee \overline{\ell}$
Pivot: ℓ or $\overline{\ell}$
Resolvent: $A \vee B = (A \vee \ell) \diamond (B \vee \overline{\ell})$

- theory solvers: theory lemmas

$$\neg(a \simeq c) \vee \neg(c \simeq b) \vee a \simeq b \qquad \neg(a \simeq b) \vee f(a) \simeq f(b)$$

$$\neg(y > 1) \vee \neg(x < 1) \vee y > x$$

Proofs in veriT

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

- $\triangleright$ SAT solver: resolution

$$\frac{A \vee \ell \qquad B \vee \overline{\ell}}{A \vee B}$$

Antecedents: $A \vee \ell$, $B \vee \overline{\ell}$
Pivot: ℓ or $\overline{\ell}$
Resolvent: $A \vee B = (A \vee \ell) \diamond (B \vee \overline{\ell})$

- $\triangleright$ theory solvers: theory lemmas

$$\neg(a \simeq c) \vee \neg(c \simeq b) \vee a \simeq b \qquad \neg(a \simeq b) \vee f(a) \simeq f(b)$$

$$\neg(y > 1) \vee \neg(x < 1) \vee y > x$$

- $\triangleright$ instantiation module: instantiation lemmas

$$\neg(\forall x.\, \psi[x]) \vee \psi[t]$$

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

- ⊖ ~~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: $(\neg \forall x.\, \mathsf{p}(x)) \simeq \neg\, \mathsf{p}(\varepsilon x.\, \neg\, \mathsf{p}(x))$

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

theory simplifications: $(\mathsf{k} + 1 \times 0 < \mathsf{k}) \simeq (\mathsf{k} < \mathsf{k})$

Inference system

A context Γ fixes a set of variables and specifies a substitution

substitution

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

bound variable

Inference system

A context Γ fixes a set of variables and specifies a substitution

$$\Gamma ::= \varnothing \mid \Gamma, \underbrace{x}_{\text{bound variable}} \mid \Gamma, \overbrace{\bar{x}_n \mapsto \bar{s}_n}^{\text{substitution}}$$

Rules have the form

$$\frac{\overbrace{\mathcal{D}_1 \quad \cdots \quad \mathcal{D}_n}^{\text{derivations of premises}}}{\underbrace{\Gamma}_{\text{assumptions}} \rhd \underbrace{t \simeq u}_{\text{transformation}}}\ \mathrm{R}$$

- $\rhd$ Semantically, the judgement expresses the equality of the terms $\Gamma(t)$ and u for all variables fixed by Γ

Example of ‘let’ expansion

$$\dfrac{\dfrac{}{\triangleright\ \mathsf{a} \simeq \mathsf{a}}\ \textsc{Cong} \qquad \dfrac{\dfrac{}{x \mapsto \mathsf{a} \triangleright x \simeq \mathsf{a}}\ \textsc{Refl} \qquad \dfrac{}{x \mapsto \mathsf{a} \triangleright x \simeq \mathsf{a}}\ \textsc{Refl}}{x \mapsto \mathsf{a} \triangleright \mathsf{p}(x, x) \simeq \mathsf{p}(\mathsf{a}, \mathsf{a})}\ \textsc{Cong}}{\triangleright\ (\text{let } x \simeq \mathsf{a} \text{ in } \mathsf{p}(x, x)) \simeq \mathsf{p}(\mathsf{a}, \mathsf{a})}\ \textsc{Let}$$

Example of theory simplification

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{}{\rhd\ \mathsf{k} \simeq \mathsf{k}}\ \textsc{Cong}
      \qquad
      \dfrac{}{\rhd\ 1 \times 0 \simeq 0}\ \textsc{Taut}_{\times}
    }{\rhd\ \mathsf{k} + 1 \times 0 \simeq \mathsf{k} + 0}\ \textsc{Cong}
    \qquad
    \dfrac{}{\rhd\ \mathsf{k} + 0 \simeq \mathsf{k}}\ \textsc{Taut}_{+}
  }{\rhd\ \mathsf{k} + 1 \times 0 \simeq \mathsf{k}}\ \textsc{Trans}
  \qquad
  \dfrac{}{\rhd\ \mathsf{k} \simeq \mathsf{k}}\ \textsc{Cong}
}{\rhd\ (\mathsf{k} + 1 \times 0 < \mathsf{k}) \simeq (\mathsf{k} < \mathsf{k})}\ \textsc{Cong}
$$

Example of skolemization

The skolemization proof of the formula $\neg \forall x.\, \mathsf{p}(x)$:

$$\dfrac{\dfrac{\dfrac{\overline{x \mapsto \varepsilon x.\, \neg \mathsf{p}(x) \,\triangleright\, x \simeq \varepsilon x.\, \neg \mathsf{p}(x)}\ \textsc{Refl}}{x \mapsto \varepsilon x.\, \neg \mathsf{p}(x) \,\triangleright\, \mathsf{p}(x) \simeq \mathsf{p}(\varepsilon x.\, \neg \mathsf{p}(x))}\ \textsc{Cong}}{\triangleright\, (\forall x.\, \mathsf{p}(x)) \simeq \mathsf{p}(\varepsilon x.\, \neg \mathsf{p}(x))}\ \textsc{Sko}_\forall}{\triangleright\, (\neg \forall x.\, \mathsf{p}(x)) \simeq \neg \mathsf{p}(\varepsilon x.\, \neg \mathsf{p}(x))}\ \textsc{Cong}$$

veriT syntax:

$$\begin{array}{l}
(.\mathrm{c0}\ (\mathsf{Sko_All}\ \ :\!\mathrm{conclusion}\ ((\forall x.\, \mathsf{p}(x)) \simeq \mathsf{p}(\varepsilon x.\, \neg \mathsf{p}(x)))\\
\qquad :\!\mathrm{args}\ (x \mapsto (\varepsilon x.\, \neg \mathsf{p}(x)))\\
\qquad :\!\mathrm{subproof}\ ((.\mathrm{c1}\ (\mathsf{Refl}\ \ :\!\mathrm{conclusion}\ (x \simeq (\varepsilon x.\, \neg \mathsf{p}(x)))))\\
\qquad\qquad\qquad (.\mathrm{c2}\ (\mathsf{Cong}\ \ :\!\mathrm{clauses}\ (.\mathrm{c1})\\
\qquad\qquad\qquad\qquad :\!\mathrm{conclusion}\ (\mathsf{p}(x) \simeq \mathsf{p}(\varepsilon x.\, \neg \mathsf{p}(x))))))))\\
(.\mathrm{c3}\ (\mathsf{Cong}\ \ :\!\mathrm{clauses}\ (.\mathrm{c0})\ \ :\!\mathrm{conclusion}\ ((\neg \forall x.\, \mathsf{p}(x)) \simeq \neg \mathsf{p}(\varepsilon x.\, \neg \mathsf{p}(x)))))
\end{array}$$

Proof-producing contextual recursion

function *process*(Δ, t)
  match t
    case x:
      return *build_var*(Δ, x)
    case $\mathsf{f}(\bar{t}_n)$:
      $\bar{\Delta}'_n \leftarrow$ (*ctx_app*(Δ, $\mathsf{f}, \bar{t}_n$, i))$_{i=1}^{n}$
      return *build_app*$\big(\Delta$, $\bar{\Delta}'_n$, f, $\bar{t}_n$, (*process*(Δ'_i, t_i))$_{i=1}^{n}\big)$
    case $Qx.\, \varphi$:
      $\Delta' \leftarrow$ *ctx_quant*(Δ, Q, x, φ)
      return *build_quant*(Δ, Δ', Q, x, φ, *process*(Δ', φ))
    case $\text{let } \bar{x}_n \simeq \bar{r}_n \text{ in } t'$:
      $\Delta' \leftarrow$ *ctx_let*(Δ, $\bar{x}_n$, $\bar{r}_n$, t')
      return *build_let*(Δ, Δ', $\bar{x}_n$, $\bar{r}_n$, t', *process*(Δ', t'))

▷ Parameterized by a notion of context and plugin functions

Theoretical properties

- Soundness of inference rules proven through an encoding into simply typed λ-calculus

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

$$\frac{\mathcal{D}_1 \qquad \cdots \qquad \mathcal{D}_n}{M \simeq N}\,\mathsf{R}$$

Theoretical properties

- ▷ Soundness of inference rules proven through an encoding into simply typed λ-calculus

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

$$\frac{\mathcal{D}_1 \quad \cdots \quad \mathcal{D}_n}{M \simeq N}\,\mathsf{R}$$

- ▷ Correctness of proof-producing contextual recursion algorithm
- ▷ Cost of proof production is linear and of proof checking is (almost) linear*

  * assuming suitable data structures

Implementation

Proof output for veriT

Framework implemented with a proof-producing contextual recursion algorithm

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

Prototype checker in Isabelle/HOL

Maps proofs into Isabelle theorems

- $\oplus$ Judgements encoded in λ-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

Summary

- ▷ 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

Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. "veriT: An Open, Trustable and Efficient SMT-Solver". In: Proc. Conference on Automated Deduction (CADE). Ed. by Renate A. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer, 2009, pp. 151–156.

David Detlefs, Greg Nelson, and James B. Saxe. "Simplify: A Theorem Prover for Program Checking". In: J. ACM 52.3 (2005), pp. 365–473.

Yeting Ge and Leonardo de Moura. "Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories". In: Computer Aided Verification (CAV). Ed. by Ahmed Bouajjani and Oded Maler. Vol. 5643. Lecture Notes in Computer Science. Springer, 2009, pp. 306–320.

Liana Hadarean, Clark W. Barrett, Andrew Reynolds, Cesare Tinelli, and Morgan Deters. "Fine Grained SMT Proofs for the Theory of Fixed-Width Bit-Vectors". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov. Vol. 9450. Lecture Notes in Computer Science. Springer, 2015, pp. 340–355.

References

Guy Katz, Clark W. Barrett, Cesare Tinelli, Andrew Reynolds, and Liana Hadarean. "Lazy proofs for DPLL(T)-based SMT solvers". In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Ruzica Piskac and Muralidhar Talupur. IEEE, 2016, pp. 93–100.

Laura Kovács and Andrei Voronkov. "First-Order Theorem Proving and Vampire". English. In: Computer Aided Verification (CAV). Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 1–35.

Leonardo de Moura and Nikolaj Bjørner. "Efficient E-Matching for SMT Solvers". In: Proc. Conference on Automated Deduction (CADE). Ed. by Frank Pfenning. Vol. 4603. Lecture Notes in Computer Science. Springer, 2007, pp. 183–198.

Leonardo Mendonça de Moura and Nikolaj Bjørner. "Proofs and Refutations, and Z3". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) Workshops. Ed. by Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, Renate A. Schmidt, and Stephan Schulz. Vol. 418. CEUR Workshop Proceedings. CEUR-WS.org, 2008.

References

Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krsti, Morgan Deters, and Clark Barrett. "Quantifier Instantiation Techniques for Finite Model Finding in SMT". In: Proc. Conference on Automated Deduction (CADE). Ed. by Maria Paola Bonacina. Vol. 7898. Lecture Notes in Computer Science. Springer, 2013, pp. 377–391.

Andrew Reynolds, Cesare Tinelli, and Leonardo Mendonça de Moura. "Finding conflicting instances of quantified formulas in SMT". In: Formal Methods In Computer-Aided Design (FMCAD). IEEE, 2014, pp. 195–202.

Stephan Schulz. "System Description: E 1.8". English. In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Ken McMillan, Aart Middeldorp, and Andrei Voronkov. Vol. 8312. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 735–743.

Geoff Sutcliffe, Jürgen Zimmer, and Stephan Schulz. "TSTP Data-Exchange Formats for Automated Theorem Proving Tools". In: Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems. Ed. by Weixiong Zhang and Volker Sorge. Vol. 112. Frontiers in Artificial Intelligence and Applications. IOS Press, 2004, pp. 201–215.

References

Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. "SPASS Version 3.5". English. In: Proc. Conference on Automated Deduction (CADE). Ed. by RenateA. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 140–145.