An introduction to SMT solving with quantifiers

Haniel Barbosa, Universidade Federal de Minas Gerais

SAT/SMT/AR Summer School

2024–06–27, LORIA–Inria, Université de Lorraine, Nancy, FR

Agenda

- What are quantifiers for in SMT?
- An overview of classic instantiation techniques
  - Trigger-based instantiation
  - Conflict-based instantiation
  - Model-based instantiation
- A unifying framework for classic instantiation techniques
- Effective enumerative instantiation
- Playing with different instantiation techniques

Why do we need quantifiers?

- We've just seen how powerful and flexible SMT solvers are.
- The efficiency of SMT solvers comes from dedicated decision procedures for their theories.
- But what if the problem you want to solve does not fit existing theories?

Example

```
(set-logic UF)

(declare-sort U 0)
(declare-fun f (U U) U)
(declare-const a U)
(declare-const b U)

(assert (not (= (f a b) (f b a))))



(check-sat)
```

Example

```
(set-logic UF)

(declare-sort U 0)
(declare-fun f (U U) U)
(declare-const a U)
(declare-const b U)

(assert (not (= (f a b) (f b a))))

(check-sat)
```

```
(set-logic UF)

(declare-sort U 0)
(declare-fun f (U U) U)
(declare-const a U)
(declare-const b U)

(assert (not (= (f a b) (f b a))))

(assert (forall ((x U) (y U)) (= (f x y) (f y x))))

(check-sat)
```

Quantifiers are important for many applications

- Automatic theorem proving
  - Adding axioms for new symbols (tools such Sledghammer [BKPU16])
- Software verification
  - Encoding contracts (tools such as Dafny [Lei10] and Verus [LHC+23] rely heavily on quantifiers)
- Function synthesis
  - Specifying the behavior of a function to synthesize [ABJ+13; RBN+19]

Quantifiers are important for many applications

- Automatic theorem proving
  - Adding axioms for new symbols (tools such Sledghammer [BKPU16])
- Software verification
  - Encoding contracts (tools such as Dafny [Lei10] and Verus [LHC+23] rely heavily on quantifiers)
- Function synthesis
  - Specifying the behavior of a function to synthesize [ABJ+13; RBN+19]
- Unfortunately, adding quantifiers leads to several complications.
  - Undecidable in general
  - Explosive heuristics
  - Users want it to work as well as on quantifier-free problems

Quantifiers are important for many applications

- Automatic theorem proving
  - Adding axioms for new symbols (tools such Sledghammer [BKPU16])
- Software verification
  - Encoding contracts (tools such as Dafny [Lei10] and Verus [LHC+23] rely heavily on quantifiers)
- Function synthesis
  - Specifying the behavior of a function to synthesize [ABJ+13; RBN+19]
- Unfortunately, adding quantifiers leads to several complications.
  - Undecidable in general
  - Explosive heuristics
  - Users want it to work as well as on quantifier-free problems
- But as we will see today, state-of-the-art solvers do well with quantifiers in practice

Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

$$\begin{array}{lll} t & ::= & x \mid f(t, \dots, t) \\ \varphi & ::= & p(t, \dots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x_1 \dots x_n .\, \varphi \end{array}$$

Given a formula φ in FOL and background theories $\mathcal{T}_1, \dots, \mathcal{T}_n$, finding a model $\mathcal{M}$ giving an *interpretation* to all terms and predicates such that $\mathcal{M} \models_{\mathcal{T}_1, \dots, \mathcal{T}_n} \varphi$.

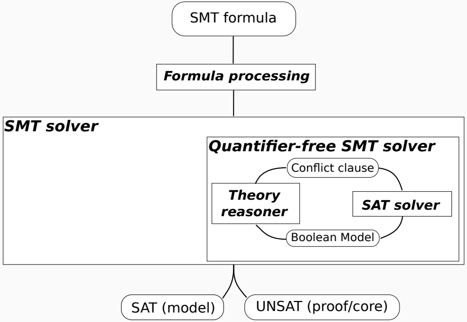
- Quantified formulas can be classified as *strong* and *weak* quantifiers, which means to occur in a negative (e.g., under a single negation) or positive context.
- It is sound to Skolemize strong quantifiers:

$$\frac{\exists x .\, \varphi[x]}{\varphi[k]} \text{ where } k \text{ is a fresh function symbol}$$

- If Skolemization is done under other quantifiers, the introduced function must take the respective quantified variables as arguments.

$$\frac{\forall y .\, \exists x .\, p(x, y)}{\forall y .\, p(f(y), y)}$$

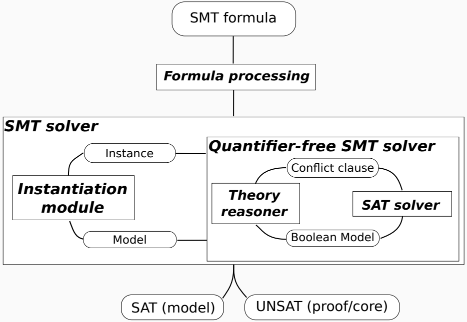
CDCL(T) architecture



Quantifier-free solver enumerates models E

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

CDCL(T) architecture



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

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

The abstract procedure: ground case

```
function CheckSat(φ, 𝒯) is
  φ ← Process(φ)                          // Simplifications, CNF transformation
  do
    E ← CheckBoolean(abs(φ))              // SAT solver
    if E = ∅ then
      return Unsat
    C ← CheckGround(E, 𝒯)                 // Theory solvers
    φ ← φ ∪ C
  while C ≠ ∅
  return Sat
```

The abstract procedure: quantified case

```
function CheckSatQ(φ, 𝒯) is
  φ ← Process(φ)                              // Simplifications, CNF transformation
  do
    ⟨E, 𝒬⟩ ← CheckBoolean(absQ(φ))            // SAT solver
    if E ∪ 𝒬 = ∅ then
      return Unsat
    C ← CheckGround(E, 𝒯)                     // Theory solvers
    if C ≠ ∅ then
      φ ← φ ∪ C
      continue
    ℐ ← Inst(E, 𝒬, 𝒯)                         // Instantiation module
    φ ← φ ∪ ℐ
  while ℐ ≠ ∅
  if models can be built for 𝒯 then
    return Sat
  else
    return Unknown
```

Why can we use instantiation?

Theorem (Herbrand)

A set of pure first-order logic formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of its instances.

Why can we use instantiation?

Theorem (Herbrand)

A set of pure first-order logic formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of its instances.

Is the following syllogism correct?

All humans are mortal
All Greeks are humans

Then all Greeks are mortal

Translate to FOL

Why can we use instantiation?

Theorem (Herbrand)

A set of pure first-order logic formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of its instances.

Is the following syllogism correct?

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

Translate to FOL

$$\frac{\forall x.\, H(x) \rightarrow M(x) \qquad \forall x.\, G(x) \rightarrow H(x)}{\forall x.\, G(x) \rightarrow M(x)}$$

Why can we use instantiation?

Theorem (Herbrand)

A set of pure first-order logic formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of its instances.

Is the following syllogism correct?

All humans are mortal
All Greeks are humans

Then all Greeks are mortal

Translate to FOL

$$\frac{\forall x.\, H(x) \rightarrow M(x) \qquad \forall x.\, G(x) \rightarrow H(x)}{\forall x.\, G(x) \rightarrow M(x)}$$

- Checking the validity of this formula:
  $\Big((\forall x.\, H(x) \rightarrow M(x)) \wedge (\forall x.\, G(x) \rightarrow H(x))\Big) \rightarrow \forall x.\, G(x) \rightarrow M(x)$

Why can we use instantiation?

Theorem (Herbrand)

A set of pure first-order logic formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of its instances.

Is the following syllogism correct?

All humans are mortal
All Greeks are humans

Then all Greeks are mortal

Translate to FOL

$$\frac{\forall x.\, H(x) \to M(x) \qquad \forall x.\, G(x) \to H(x)}{\forall x.\, G(x) \to M(x)}$$

- Checking the validity of this formula:
$$\Big((\forall x.\, H(x) \to M(x)) \land (\forall x.\, G(x) \to H(x)) \Big) \to \forall x.\, G(x) \to M(x)$$
- Checking the unsatisfiability of:
$$\forall x.\, H(x) \to M(x), \forall x.\, G(x) \to H(x), \neg(\forall x.\, G(x) \to M(x))$$

Why can we use instantiation?

Theorem (Herbrand)

A set of pure first-order logic formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of its instances.

Is the following syllogism correct?

All humans are mortal
All Greeks are humans

Then all Greeks are mortal

Translate to FOL

$$\frac{\forall x.\, H(x) \to M(x) \qquad \forall x.\, G(x) \to H(x)}{\forall x.\, G(x) \to M(x)}$$

- Checking the validity of this formula:
  $\Big(\big(\forall x.\, H(x) \to M(x)\big) \wedge \big(\forall x.\, G(x) \to H(x)\big)\Big) \to \forall x.\, G(x) \to M(x)$
- Checking the unsatisfiability of:
  $\forall x.\, H(x) \to M(x), \forall x.\, G(x) \to H(x), \neg(\forall x.\, G(x) \to M(x))$
- Skolemize: $\forall x.\, H(x) \to M(x), \forall x.\, G(x) \to H(x), G(s), \neg M(s)$

Why can we use instantiation?

Theorem (Herbrand)

A set of pure first-order logic formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of its instances.

Is the following syllogism correct?

All humans are mortal
All Greeks are humans

Then all Greeks are mortal

Translate to FOL

$$\frac{\forall x.\, H(x) \to M(x) \qquad \forall x.\, G(x) \to H(x)}{\forall x.\, G(x) \to M(x)}$$

- Checking the validity of this formula:
  $\Big(\big(\forall x.\, H(x) \to M(x)\big) \land \big(\forall x.\, G(x) \to H(x)\big)\Big) \to \forall x.\, G(x) \to M(x)$
- Checking the unsatisfiability of:
  $\forall x.\, H(x) \to M(x), \forall x.\, G(x) \to H(x), \neg(\forall x.\, G(x) \to M(x))$
- Skolemize: $\forall x.\, H(x) \to M(x), \forall x.\, G(x) \to H(x), G(s), \neg M(s)$
- Instantiate: add the two formulas $H(s) \to M(s), G(s) \to H(s)$

Why can we use instantiation?

Theorem (Herbrand)

A set of pure first-order logic formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of its instances.

Is the following syllogism correct?

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

Translate to FOL

$$\frac{\forall x.\, H(x) \rightarrow M(x) \qquad \forall x.\, G(x) \rightarrow H(x)}{\forall x.\, G(x) \rightarrow M(x)}$$

- Checking the validity of this formula:
  $\Big(\big(\forall x.\, H(x) \rightarrow M(x)\big) \land \big(\forall x.\, G(x) \rightarrow H(x)\big)\Big) \rightarrow \forall x.\, G(x) \rightarrow M(x)$
- 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: $\forall x.\, H(x) \rightarrow M(x), \forall x.\, G(x) \rightarrow H(x), G(s), \neg M(s)$
- Instantiate: add the two formulas $H(s) \rightarrow M(s), G(s) \rightarrow H(s)$
- A ground SMT solver will deduce unsatisfiability.

Instantiation is not the only way to reason about first-order logic

- Superposition-based, tableaux-based systems are well-established theorem provers
  - Vampire [KV13], E [SCV19], ...
  - Princess [BR15], LeanCop [Ott08], ...

- The focus on instantiation in SMT can be explained by how it makes “quantifier reasoning” simulate how the other theory solvers work, which is well-suited for the CDCL(T) architecture.

Instantiation techniques

- Trigger-based [DNS05; MB07]
- Conflict-based [RTM14; BFR17]
- Model-based [GM09; RTG+13]

⊕ General: ∀+EUF+...

⊖ Finding instantiations is hard

- Enumerative [RBF18]

⊕ Easy to implement

⊕ Reliable last resort

- QE-based [Mon10; Bjø10; RDK+15; BJ15]

⊕ Decision procedures available

⊖ Pure fragments

- Syntax-Guided Synthesis (SyGuS)-based [NPR+21]

⊕ Covers pure theories where QE is not easily available

⊖ Very expensive

Instantiation techniques

- Trigger-based [DNS05; MB07]
- Conflict-based [RTM14; BFR17]
- Model-based [GM09; RTG+13]

⊕ General: $\forall$+EUF+...

⊖ Finding instantiations is hard

- Enumerative [RBF18]

⊕ Easy to implement

⊕ Reliable last resort

- QE-based [Mon10; Bjø10; RDK+15; BJ15]

⊕ Decision procedures available

⊖ Pure fragments

- Syntax-Guided Synthesis (SyGuS)-based [NPR+21]

⊕ Covers pure theories where QE is not easily available

⊖ Very expensive

Instantiation strategies: trigger-based

[Detlefs et al. J. ACM'05]

Trigger-based instantiation (E-matching): search for relevant instantiations according to a set of triggers and E-matching

- A *trigger* is a set of terms whose free variables should cover the respective quantified variables.

Instantiation strategies: trigger-based

[Detlefs et al. J. ACM'05]

Trigger-based instantiation (E-matching): search for relevant instantiations according to a set of triggers and E-matching

- A *trigger* is a set of terms whose free variables should cover the respective quantified variables.

- E $= \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and Q $= \{\forall x.\, P(x) \vee R(x)\}$
- Assume the trigger $\{(P(x))\}$.
- Since E $\models P(x)\{x \mapsto t\} \simeq P(t)$, for $t = a, b, c$, this strategy may return $\{\{x \mapsto a\},\ \{x \mapsto b\},\ \{x \mapsto c\}\}$.
- Formally:

$\mathbf{e}$(E, $\forall \bar{x}.\, \varphi$):

1. Select a trigger $\{\bar{t}_1, \ldots \bar{t}_n\}$ for $\forall \bar{x}.\, \varphi$.
2. For each $i = 1, \ldots, n$, select a set of substitutions S_i s.t. for each $\sigma \in S_i$, E $\models \bar{t}_i\sigma \simeq \bar{g}_i$ for some tuple $\bar{g}_i \in \mathbf{T}$(E).
3. Return $\bigcup_{i=1}^{n} S_i$.

Trigger-based instantiation is highly dependent on the chosen triggers

- A proper selection of triggers may guarantee a decision procedure for some fragments [DCKP13].
- But in general, trigger selection can have a high impact on the solver's success rate
- Again for E $= \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and Q $= \{\forall x.\, P(x) \vee R(x)\}$
- Assume the trigger $\{(R(x))\}$.
- Now there is only one possible instantiation: $\{x \mapsto b\}$

A trigger selection strategy [LP16]

1. Traverse formula and collect *trigger heads* and *trigger killers*.
   - *Trigger heads* contain at least one variable and *no* trigger killers
   - *Trigger killers* are typically applications of interpreted symbols (arithmetic, logical connectives, ...).

A trigger selection strategy [LP16]

1. Traverse formula and collect *trigger heads* and *trigger killers*.
   - *Trigger heads* contain at least one variable and *no* trigger killers
   - *Trigger killers* are typically applications of interpreted symbols (arithmetic, logical connectives, ...).
2. Candidate triggers are built by combining trigger heads while ensuring two properties: *adequacy* and *parsimony*.
   - A candidate is adequate if it contains all the variables
   - It is parsimonious if removing any term from the candidate makes it inadequate.

A trigger selection strategy [LP16]

1. Traverse formula and collect *trigger heads* and *trigger killers*.
   - *Trigger heads* contain at least one variable and *no* trigger killers
   - *Trigger killers* are typically applications of interpreted symbols (arithmetic, logical connectives, ...).
2. Candidate triggers are built by combining trigger heads while ensuring two properties: *adequacy* and *parsimony*.
   - A candidate is adequate if it contains all the variables
   - It is parsimonious if removing any term from the candidate makes it inadequate.
3. For the formula $\forall x.\, p_1(x) \vee \cdots \vee p_n(x)$ we have:
   - Trigger heads: $\{p_1(x), \ldots, p_n(x)\}$
   - 2^n adequate candidates
   - n which are parsimonious: the singletons $\{p_1(x)\}, \ldots, \{p_n(x)\}$.

A trigger selection strategy [LP16]

1. Traverse formula and collect *trigger heads* and *trigger killers*.
   - *Trigger heads* contain at least one variable and *no* trigger killers
   - *Trigger killers* are typically applications of interpreted symbols (arithmetic, logical connectives, ...).
2. Candidate triggers are built by combining trigger heads while ensuring two properties: *adequacy* and *parsimony*.
   - A candidate is adequate if it contains all the variables
   - It is parsimonious if removing any term from the candidate makes it inadequate.
3. For the formula $\forall x.\, p_1(x) \vee \cdots \vee p_n(x)$ we have:
   - Trigger heads: $\{p_1(x), \ldots, p_n(x)\}$
   - 2^n adequate candidates
   - n which are parsimonious: the singletons $\{p_1(x)\}, \ldots, \{p_n(x)\}$.
4. Remaining candidates ordered by specificity:
   - T_1 is less *specific* than T_2 if and only if all matchings of T_2 are also matchings for T_1
   - For each t in T_1 there is a trigger head t' in T_2 such that t matches t', i.e. there is a substitution σ such that $t\sigma = t'$.

A trigger selection strategy [LP16]

1. Traverse formula and collect *trigger heads* and *trigger killers*.
   - *Trigger heads* contain at least one variable and *no* trigger killers
   - *Trigger killers* are typically applications of interpreted symbols (arithmetic, logical connectives, ...).
2. Candidate triggers are built by combining trigger heads while ensuring two properties: *adequacy* and *parsimony*.
   - A candidate is adequate if it contains all the variables
   - It is parsimonious if removing any term from the candidate makes it inadequate.
3. For the formula $\forall x.\, p_1(x) \vee \cdots \vee p_n(x)$ we have:
   - Trigger heads: $\{p_1(x), \ldots, p_n(x)\}$
   - 2^n adequate candidates
   - n which are parsimonious: the singletons $\{p_1(x)\}, \ldots, \{p_n(x)\}$.
4. Remaining candidates ordered by specificity:
   - T_1 is less *specific* than T_2 if and only if all matchings of T_2 are also matchings for T_1
   - For each t in T_1 there is a trigger head t' in T_2 such that t matches t', i.e. there is a substitution σ such that $t\sigma = t'$.
5. Finally, the possible triggers for the quantified formula are the minimal candidates.

Trigger-based instantiation issues: matching loops

- A well known issue is matching loops: terms from previous instantiation rounds leading to more instantiations indefinitely

- consider E $= \{a \simeq f(a), \dots\}$ and Q $= \{\forall x.\, f(f(x)) \simeq f(x)\}$. What happens if the trigger is $\{f(x)\}$?

Trigger-based instantiation issues: matching loops

- A well known issue is matching loops: terms from previous instantiation rounds leading to more instantiations indefinitely
- consider E $= \{a \simeq f(a), \ldots\}$ and Q $= \{\forall x.\, f(f(x)) \simeq f(x)\}$. What happens if the trigger is $\{f(x)\}$?
  - E $= \{a \simeq f(a), f(a) \simeq f(f(a)), \ldots\}$

Trigger-based instantiation issues: matching loops

- A well known issue is matching loops: terms from previous instantiation rounds leading to more instantiations indefinitely

- consider E $= \{a \simeq f(a), \dots\}$ and Q $= \{\forall x.\ f(f(x)) \simeq f(x)\}$. What happens if the trigger is $\{f(x)\}$?
  - E $= \{a \simeq f(a), f(a) \simeq f(f(a)), \dots\}$
  - E $= \{a \simeq f(a), f(a) \simeq f(f(a)), f(f(a)) \simeq f(f(f(a))), \dots\}$

Trigger-based instantiation issues: matching loops

- A well known issue is matching loops: terms from previous instantiation rounds leading to more instantiations indefinitely

- consider E $= \{a \simeq f(a), \dots\}$ and Q $= \{\forall x.\ f(f(x)) \simeq f(x)\}$. What happens if the trigger is $\{f(x)\}$?
  - E $= \{a \simeq f(a), f(a) \simeq f(f(a)), \dots\}$
  - E $= \{a \simeq f(a), f(a) \simeq f(f(a)), f(f(a)) \simeq f(f(f(a))), \dots\}$

- Some approaches introduced to address this issue, but they have limited application:
  - Select triggers such that they are not subterms of other terms in the formula [LP16]
  - Ignore during instantiation terms from instances that did not lead to conflicts [Bar16; MB07]

Trigger-based instantiation can be explosive

Pattern-matching of terms from Q into terms of E

- for $\forall xyz.\ f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ a trigger is $\{f(x),\ g(y),\ h(z)\}$

⊖ Multi-term triggers specially can lead to many instantiations

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

Trigger-based instantiation can be explosive

Pattern-matching of terms from Q into terms of E

- for $\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ a trigger is $\{f(x),\, g(y),\, h(z)\}$

⊖ Multi-term triggers specially can lead to many instantiations

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

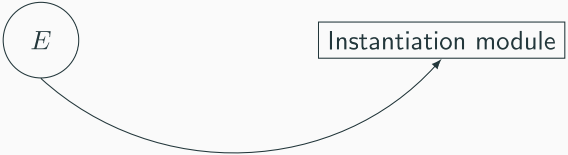
Trigger-based instantiation can be explosive

Pattern-matching of terms from Q into terms of E

- for $\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ a trigger is $\{f(x),\, g(y),\, h(z)\}$

⊖ Multi-term triggers specially can lead to many instantiations

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



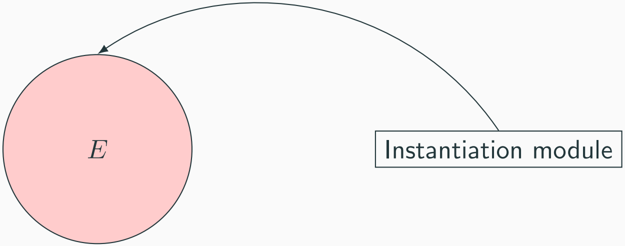
Trigger-based instantiation can be explosive

Pattern-matching of terms from Q into terms of E

- for $\forall xyz.\ f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ a trigger is $\{f(x),\ g(y),\ h(z)\}$

⊖ Multi-term triggers specially can lead to many instantiations

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



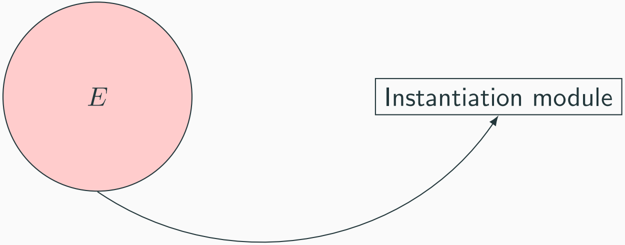
Trigger-based instantiation can be explosive

Pattern-matching of terms from Q into terms of E

- for $\forall xyz.\, f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ a trigger is $\{f(x),\, g(y),\, h(z)\}$

⊖ Multi-term triggers specially can lead to many instantiations

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



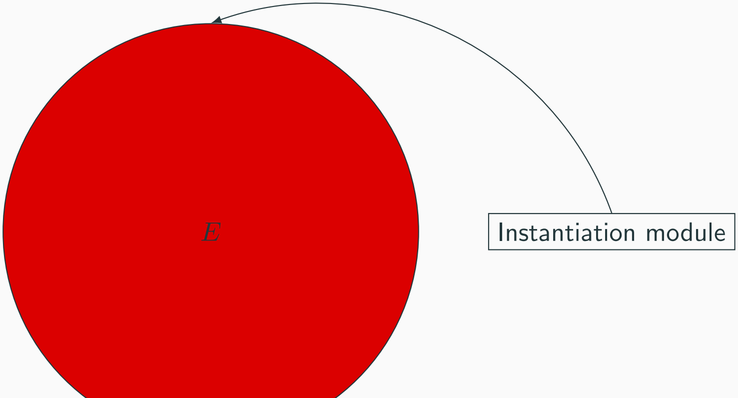
Trigger-based instantiation can be explosive

Pattern-matching of terms from Q into terms of E

- for $\forall xyz.\ f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ a trigger is $\{f(x),\ g(y),\ h(z)\}$

⊖ Multi-term triggers specially can lead to many instantiations

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



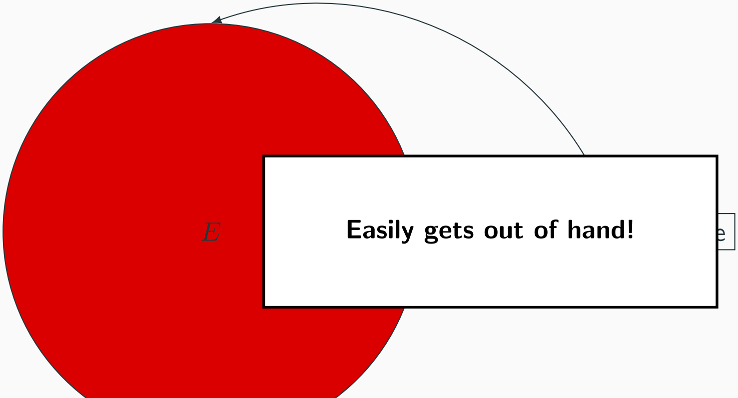
Trigger-based instantiation can be explosive

Pattern-matching of terms from Q into terms of E

- for $\forall xyz.\ f(x) \not\simeq f(z) \vee g(y) \simeq h(z)$ a trigger is $\{f(x),\ g(y),\ h(z)\}$

⊖ Multi-term triggers specially can lead to many instantiations

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



Goal-oriented instantiation

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

⊕ Only instances refuting the current model are generated

Goal-oriented instantiation

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

⊕ 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

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

Goal-oriented instantiation

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

- ⊕ 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

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

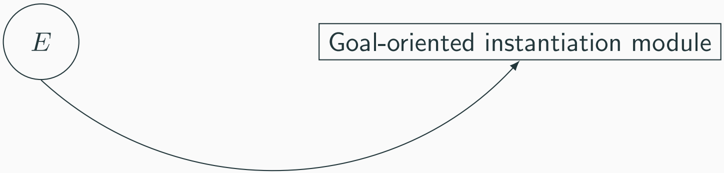
Goal-oriented instantiation

Check consistency of $E \cup Q$

⊕ 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

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



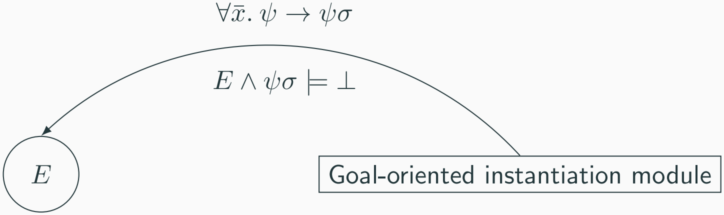
Goal-oriented instantiation

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

⊕ 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

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



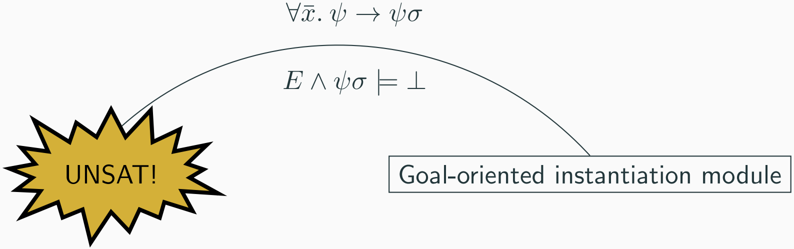
Goal-oriented instantiation

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

- ⊕ 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

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



Instantiation strategies: conflict-based [Reynolds et al. FMCAD'14]

Conflict-based instantiation: search for instantiations of a quantified formula in Q that make E unsatisfiable

Instantiation strategies: conflict-based

[Reynolds et al. FMCAD'14]

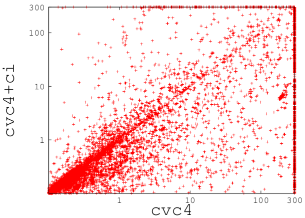
Conflict-based instantiation: search for instantiations of a quantified formula in Q that make E unsatisfiable

- E $= \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and Q $= \{\forall x.\, P(x) \vee R(x)\}$
- Since E, $P(b) \vee R(b) \models \bot$, this strategy will return $\{\{x \mapsto b\}\}$.
- Formally:

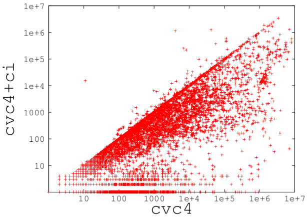
$\mathbf{c}$(E, $\forall \bar{x}.\, \varphi$): 1. Either return $\{\sigma\}$ where E, $\varphi\sigma \models \bot$, or return $\emptyset$.

Impact of conflict-based instantiation

[Reynolds et al. FMCAD'14]

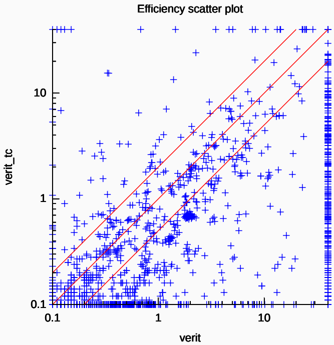


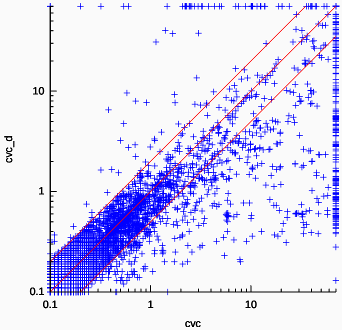
(a) Runtime (in seconds).



(b) Reported number of instances.

* experiments across 12468 benchmarks from TPTP, SMT-LIB, and Sledghammer.





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

CVC4: $+$ 200 out of 745 unsolved problems

- Improvements on CVC4 came from discarding from trigger-based strategy instances already entailed by the formula: if $E \models \varphi[\bar{t}]$, for $\forall x.\, \varphi[x]$.

* experiments in the "UF", "UFLIA", "UFLRA" and "UFIDL" categories of SMT-LIB, which have $10\,495$ benchmarks annotated as unsatisfiable, with 30s timeout.

Caveats of conflct-based instantiation

- The search for the instantiations in practice is more expensive

- The technique is inherently incomplete: it only considers a single formula at a time
  - $E = \{p(a)\}$ and Q $= \{\forall x.\, q(x), \forall yz.\, \neg q(y) \vee \neg p(z)\}$.
    There are no substitutions σ, ρ such that $E \models \neg q(x)\sigma$ or $E \models q(y)\rho \wedge p(z)\rho$, even though $E \cup$ Q is clearly inconsistent.

- It should be seen as a *complement* to other techniques that are more general

Instantiation strategies: model-based

[Ge and de Moura CAV'09]

Model-based instantiation (MBQI): build a candidate model for $E \cup Q$ and instantiate with counter-examples from model checking

Instantiation strategies: model-based

[Ge and de Moura CAV'09]

Model-based instantiation (MBQI): build a candidate model for $\mathsf{E} \cup \mathsf{Q}$ and instantiate with counter-examples from model checking

- $\mathsf{E} = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $\mathsf{Q} = \{\forall x.\, P(x) \vee R(x)\}$
- Assume that $P^{\mathcal{M}} = \lambda x.\, \mathsf{ite}(x \simeq c,\ \top,\ \bot)$ and $R^{\mathcal{M}} = \lambda x.\, \bot$.
- Since $\mathcal{M} \not\models P(a) \vee R(a)$, this strategy may return $\{\{x \mapsto a\}\}$.
- Formally:

$\mathbf{m}(\mathsf{E}, \forall \bar{x}.\, \varphi)$: 1. Construct a model $\mathcal{M}$ for E.

2. Return $\{\{\bar{x} \mapsto \bar{t}\}\}$ where $\bar{t} \in \mathbf{T}(\mathsf{E})$ and $\mathcal{M} \not\models \varphi\{\bar{x} \mapsto \bar{t}\}$, or $\emptyset$ if none exists.

Instantiation strategies: model-based

[Ge and de Moura CAV'09]

- MBQI is complete for a number of fragments
  - Bernays-Schönfinkel
  - *Essentially Uninterpreted Formulas*: "theory variables" only appear as arguments of uninterpreted functions
- It is a good strategy to complement incomplete techniques
- One should note that by its nature it is generally more successful on satisfiable problems

A unifying framework for classic instantiation techniques

Let's look deeper into the problem (with $T =$ EUF)

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

Let's look deeper into the problem (with $T =$ EUF)

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

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

Let's look deeper into the problem (with $T = \text{EUF}$)

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

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

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

Let's look deeper into the problem (with $T = \mathsf{EUF}$)

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

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

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

- Each literal in the right hand side delimits possible σ

Let's look deeper into the problem (with $T = \text{EUF}$)

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

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

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

- 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 (with $T = \text{EUF}$)

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

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

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

- 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 (with $T = \text{EUF}$)

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

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

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

- 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 (with $T = \text{EUF}$)

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

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

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

- 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 (with $T = \text{EUF}$)

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

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

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

- 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 [BFR17]

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

$$s_1\sigma \simeq t_1\sigma, \ldots, s_n\sigma \simeq t_n\sigma \models u\sigma \simeq v\sigma$$

Casting instantiation techniques: Trigger-based

$$E \models (u_1 \simeq y_1 \wedge \cdots \wedge u_m \simeq y_m)\,\sigma$$

where $\{u_1, \ldots, u_m\}$ is a trigger for $\forall \bar{x}.\, \psi \in$ Q and each $y_i\sigma \in \mathbf{T}(E)$

- - $E = \{f(a) \simeq g(b),\ h(a) \simeq b,\ f(a) \simeq f(c)\}$
  - Q $= \{\forall x.\ f(x) \not\simeq g(h(x))\}$, Trigger$=\{f(x)\}$
- Solving $E \models (f(x) \simeq y)\sigma$ yields
  - $\sigma_1 = \{y \mapsto f(a),\, x \mapsto a\}$
  - $\sigma_2 = \{y \mapsto f(c),\, x \mapsto c\}$
- The instantiation lemmas are:
  - $\forall x.\ f(x) \not\simeq g(h(x)) \rightarrow f(a) \not\simeq g(h(a))$
  - $\forall x.\ f(x) \not\simeq g(h(x)) \rightarrow f(c) \not\simeq g(h(c))$

Casting instantiation techniques: Conflict-based

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

- Consider
  - $E = \{f(a) \simeq g(b),\ h(a) \simeq b,\ f(a) \simeq f(c)\}$
  - $\mathsf{Q} = \{\forall x.\ f(x) \not\simeq g(h(x))\}$
- Solving $E \models (f(x) \simeq g(h(x)))\sigma$ yields
  - $\sigma = \{x \mapsto a\}$
- The instantiation lemma is:
  - $\forall x.\ f(x) \not\simeq g(h(x)) \rightarrow f(a) \not\simeq g(h(a))$

Casting instantiation techniques: Model-based

$$E_{\text{TOT}} \models \neg\psi\sigma, \text{ for some } \forall\bar{x}.\,\psi \in \mathsf{Q}$$

where E_{TOT} is a total extension of E s.t.:

- ground terms not in E necessary for evaluating Q are added
- all terms in $\mathbf{T}(E)$ not asserted equal are made disequal

- Consider
  - $E = \{f(a) \simeq g(b),\ h(a) \simeq b\}$
  - $\mathsf{Q} = \{\forall x.\ f(x) \not\simeq g(x),\ \forall xy.\ \psi\}$, $e = a$ as a default value, and

$$\begin{aligned} E_{\text{TOT}} &= E \cup \quad \{a \not\simeq b, a \not\simeq f(a), b \not\simeq f(a)\} \\ &\qquad\ \cup \quad \{f(b) \simeq f(a), f(f(a)) \simeq f(a), g(a) \simeq a, g(f(a)) \simeq a\} \cup \{\dots\} \end{aligned}$$

- Solving $\{\dots, f(a) \simeq g(b), f(b) \simeq f(a), \dots\} \models f(x) \simeq g(x)\sigma$ yields
  - $\sigma = \{x \mapsto b\}$
- The lemma $\forall x.\ f(x) \not\simeq g(x) \rightarrow f(a) \not\simeq g(a)$ prevents the same E_{TOT}

How to solve the E-ground (dis)unification problem?

Entailment conditions:

- $E \models (x \simeq y)\sigma$
  - $x\sigma = y\sigma$ or
  - some t_1, t_2 s.t. $x\sigma \in [t_1]$, $y\sigma \in [t_2]$, and $[t_1] = [t_2]$

How to solve the E-ground (dis)unification problem?

Entailment conditions:

- $E \models (x \simeq y)\sigma$
  - $x\sigma = y\sigma$ or
  - some t_1, t_2 s.t. $x\sigma \in [t_1]$, $y\sigma \in [t_2]$, and $[t_1] = [t_2]$
- $E \models (x \simeq f(s_1, \dots, s_n))\sigma$, x occurs in $f(s_1, \dots, s_n)$,
  - some t_1, $t_2 \in \mathbf{T}(E)$ s.t. $x\sigma \in [t_1]$, $f(s_1, \dots, s_n)\sigma \in [t_2]$, and $[t_1] = [t_2]$

How to solve the E-ground (dis)unification problem?

Entailment conditions:

- $E \models (x \simeq y)\sigma$
  - $x\sigma = y\sigma$ or
  - some t_1, t_2 s.t. $x\sigma \in [t_1]$, $y\sigma \in [t_2]$, and $[t_1] = [t_2]$
- $E \models (x \simeq f(s_1, \ldots, s_n))\sigma$, x occurs in $f(s_1, \ldots, s_n)$,
  - some t_1, $t_2 \in \mathbf{T}(E)$ s.t. $x\sigma \in [t_1]$, $f(s_1, \ldots, s_n)\sigma \in [t_2]$, and $[t_1] = [t_2]$
- $E \models (x \simeq f(s_1, \ldots, s_n))\sigma$, x does not occur in $f(s_1, \ldots, s_n)$ and
  - $x\sigma = f(s_1, \ldots, s_n)\sigma$ or
  - some t_1, t_2 s.t. $x\sigma \in [t_1]$, $f(s_1, \ldots, s_n)\sigma \in [t_2]$, and $[t_1] = [t_2]$

How to solve the E-ground (dis)unification problem?

Entailment conditions:

- $E \models (x \simeq y)\sigma$
  - $x\sigma = y\sigma$ or
  - some t_1, t_2 s.t. $x\sigma \in [t_1]$, $y\sigma \in [t_2]$, and $[t_1] = [t_2]$
- $E \models (x \simeq f(s_1, \ldots, s_n))\sigma$, x occurs in $f(s_1, \ldots, s_n)$,
  - some t_1, $t_2 \in \mathbf{T}(E)$ s.t. $x\sigma \in [t_1]$, $f(s_1, \ldots, s_n)\sigma \in [t_2]$, and $[t_1] = [t_2]$
- $E \models (x \simeq f(s_1, \ldots, s_n))\sigma$, x does not occur in $f(s_1, \ldots, s_n)$ and
  - $x\sigma = f(s_1, \ldots, s_n)\sigma$ or
  - some t_1, t_2 s.t. $x\sigma \in [t_1]$, $f(s_1, \ldots, s_n)\sigma \in [t_2]$, and $[t_1] = [t_2]$
- $E \models (f(u_1, \ldots, u_n) \simeq g(v_1, \ldots, v_n))\sigma$ and
  - $f = g$ and $E \models u_1\sigma \simeq v_1\sigma$, ..., $E \models u_n\sigma \simeq v_n\sigma$ or
  - some t_1, $t_2 \in \mathbf{T}(E)$ s.t. $[t_1] = [t_2]$, $f(u_1, \ldots, u_n)\sigma \in [t_1]$, and $g(v_1, \ldots, v_n)\sigma \in [t_2]$

How to solve the E-ground (dis)unification problem?

Entailment conditions:

- $E \models (x \simeq y)\sigma$
  - $x\sigma = y\sigma$ or
  - some t_1, t_2 s.t. $x\sigma \in [t_1]$, $y\sigma \in [t_2]$, and $[t_1] = [t_2]$
- $E \models (x \simeq f(s_1, \ldots, s_n))\sigma$, x occurs in $f(s_1, \ldots, s_n)$,
  - some t_1, $t_2 \in \mathbf{T}(E)$ s.t. $x\sigma \in [t_1]$, $f(s_1, \ldots, s_n)\sigma \in [t_2]$, and $[t_1] = [t_2]$
- $E \models (x \simeq f(s_1, \ldots, s_n))\sigma$, x does not occur in $f(s_1, \ldots, s_n)$ and
  - $x\sigma = f(s_1, \ldots, s_n)\sigma$ or
  - some t_1, t_2 s.t. $x\sigma \in [t_1]$, $f(s_1, \ldots, s_n)\sigma \in [t_2]$, and $[t_1] = [t_2]$
- $E \models (f(u_1, \ldots, u_n) \simeq g(v_1, \ldots, v_n))\sigma$ and
  - $f = g$ and $E \models u_1\sigma \simeq v_1\sigma, \ldots, E \models u_n\sigma \simeq v_n\sigma$ or
  - some t_1, $t_2 \in \mathbf{T}(E)$ s.t. $[t_1] = [t_2]$, $f(u_1, \ldots, u_n)\sigma \in [t_1]$, and $g(v_1, \ldots, v_n)\sigma \in [t_2]$
- $E \models (u \not\simeq v)\sigma$
  - some t_1, $t_2 \in \mathbf{T}(E)$ s.t. $E \models t_1 \not\simeq t_2$, $u\sigma \in [t_1]$, and $v\sigma \in [t_2]$

Congruence Closure with Free Variables

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

⊕ (allows for) Goal-oriented instantiation technique

⊕ Efficient

⊖ ~~Ad-hoc~~ **Versatile framework, recasting many instantiation techniques as a CCFV problem**

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 $E \models L\sigma$

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

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

$\emptyset$

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

$y \simeq b$

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

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

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

$x \simeq a$ | $x \simeq c$

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

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

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

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

$$\varnothing$$

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

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

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

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

$\varnothing$

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

$y \simeq b$

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

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

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

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

$\varnothing$

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

$y \simeq b$

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

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

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

$x \simeq a$ | $x \simeq c$

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

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

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

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

$\varnothing$

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

$y \simeq b$

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

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

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

$x \simeq a$ | $x \simeq c$

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

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

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

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

$\varnothing$

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

$y \simeq b$

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

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

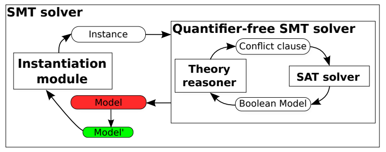
$$f(x) \simeq f(c)$$

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

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

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

Implementation



- Model minimization
  - Relevancy
  - Prime implicant
- Top symbol indexing of E-graph from ground congruence closure

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

  - $E \models f(x)\sigma \simeq t$ only if $[t]$ contains some $f(t')$
    $E \models f(x)\sigma \simeq g(y)\sigma$ only if some $[t]$ contains some $f(t')$ and some $g(t'')$
    - Bitmasks for fast checking if symbol has applications in congruence class
- Mapping from congruence class to classes it's disequal to

Implementation

- Selection strategies

$$E \models f(x, y) \simeq h(z) \land x \simeq t \land \dots$$

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

$$E \models f(t, y) \simeq h(z) \land \dots$$

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

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

Effective enumerative instantiation

Instantiation and the Herbrand Theorem

- The earliest theorem provers relied on *Herbrand instantiation*
  - Instantiate with all possible terms in the Herbrand universe (all possible well-sorted ground terms in the formula's signature)
- Enumerating all instances is unfeasible in practice!
- Enumerative instantiation was then discarded

Instantiation and the Herbrand Theorem

- The earliest theorem provers relied on *Herbrand instantiation*
  - Instantiate with all possible terms in the Herbrand universe (all possible well-sorted ground terms in the formula's signature)

- Enumerating all instances is unfeasible in practice!

- Enumerative instantiation was then discarded

But enumerative instantiation can be effective for state-of-the-art SMT

- strengthened version of the Herbrand theorem
- efficient implementation techniques

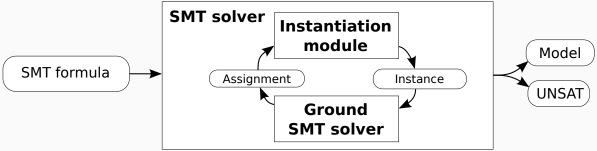
Theorem (Strengthened Herbrand [Kan63; RBF18])

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

- $\mathsf{Q}_i = \{\varphi\sigma \mid \forall \bar{x}.\, \varphi \in \mathsf{Q},\ \mathsf{dom}(\sigma) = \{\bar{x}\} \wedge \mathsf{ran}(\sigma) \subseteq \mathbf{T}(\mathsf{E}_i)\}$;
- $\mathsf{E}_0 = \mathsf{E}$, $\mathsf{E}_{i+1} \models \mathsf{E}_i \cup \mathsf{Q}_i$;

then $\mathsf{E} \cup \mathsf{Q}$ *is satisfiable in the empty theory with equality.*

Direct application at



- Quantifier-free solver enumerates assignments $\mathsf{E} \cup \mathsf{Q}$
- Instantiation module generates instances of Q

Enumerative instantiation

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

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

- $\mathsf{E} = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $\mathsf{Q} = \{\forall x.\, P(x) \vee R(x)\}$
- $\mathbf{u}$ chooses an ordering on tuples of terms, say the lexicographic extension of $\preceq$ where $a \prec b \prec c$.
- Since E does not entail $P(a) \vee R(a)$, this strategy returns $\{\{x \mapsto a\}\}$.

u as an alternative for **m**

- Enumerative instantiation plays a similar role to MBQI
- It can also serve as a "completeness fallback" to **c** and **e**
- However, **u** has advantages over **m** for UNSAT problems
- Moreover it is significantly simpler to implement
  - No model building
  - No model checking

Example

$$\mathsf{E} = \{\neg P(a),\ R(b),\ S(c)\}$$
$$\mathsf{Q} = \{\forall x.\ R(x) \vee S(x),\ \forall x.\ \neg R(x) \vee P(x),\ \forall x.\ \neg S(x) \vee P(x)\}$$
$$\mathsf{M} = \left\{ \begin{array}{lcl} P^{\mathcal{M}} & = & \lambda x.\ \bot, \\ R^{\mathcal{M}} & = & \lambda x.\ \mathsf{ite}(x \simeq b,\ \top,\ \bot), \\ S^{\mathcal{M}} & = & \lambda x.\ \mathsf{ite}(x \simeq c,\ \top,\ \bot) \end{array} \right\}, \qquad a \prec b \prec c$$

φ	x s.t. $\mathcal{M} \not\models \varphi$	x s.t. $\mathsf{E} \not\models \varphi$	$\mathbf{m}(\mathsf{E}, \forall x.\ \varphi)$	$\mathbf{u}(\mathsf{E}, \forall x.\ \varphi)$
$R(x) \vee S(x)$	a	a	$\{\{x \mapsto a\}\}$	$\{\{x \mapsto a\}\}$
$\neg R(x) \vee P(x)$	b	a, b, c	$\{\{x \mapsto b\}\}$	$\{\{x \mapsto a\}\}$
$\neg S(x) \vee P(x)$	c	a, b, c	$\{\{x \mapsto c\}\}$	$\{\{x \mapsto a\}\}$

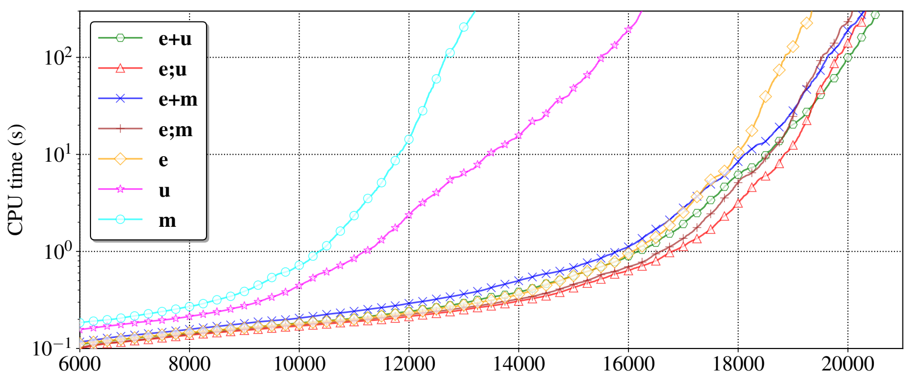
- **u** instantiates uniformly so that new terms are introduced less often
- **m** instantiates depending on how model was built
- Moreover, **u** leads to $\mathsf{E} \wedge \mathsf{Q}\{x \mapsto a\} \models \bot$
- **m** requires considering E' which satisfies E along the new instances

Implementation

Implementing enumerative instantiation efficiently depends on:

- Restricting enumeration space
- Avoiding entailed instantiations
- Term ordering to introduce new terms less often

CVC4 configurations on unsatisfiable benchmarks



- 42 065 benchmarks, being 14 731 from TPTP and 27 334 from SMT-LIB
- **e**+**u** stands for "interleave **e** and **u**", while **e**;**u** for "apply **e** first, then **u** if it fails"
- All CVC4 configurations have "**c**;" as prefix

Impact of u on satisfiable benchmarks

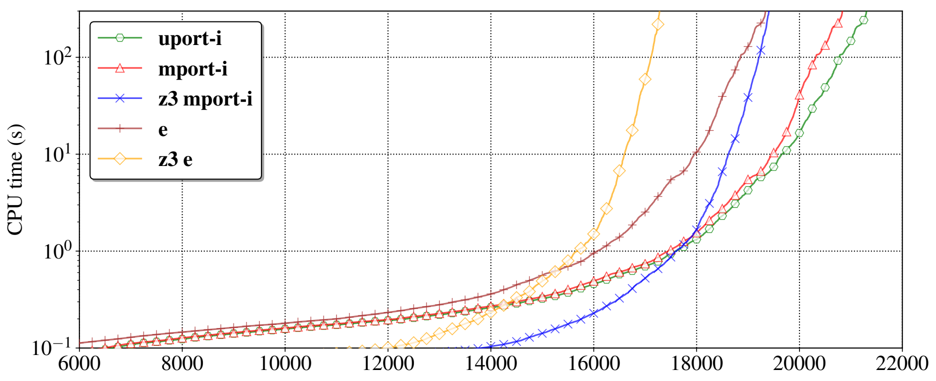
Library	#	**u**	**e;u**	**e+u**	**e**	**m**	**e;m**	**e+m**
TPTP	14731	471	492	464	17	930	808	829
UF	7293	39	42	42	0	70	69	65
Theories	20041	3	3	3	3	350	267	267
Total	42065	513	537	509	20	1350	1144	1161

- As expected, **m** greatly outperforms **u**
- Nevertheless **u** answers SAT *half as often* as **m** in empty theory
- Moreover, **u** solves 13 problems **m** does not

Impact of u on unsatisfiable benchmarks

- **u** solves $3\,043$ more benchmarks than **m**
- **u** solves $1\,737$ problems not solvable by **e**
- Combinations of **e** with **u** or **m** lead to significant gains
- **e+u** is best configuration, solving 253 more problems than **e+m** and $1\,295$ more than **e**
- Some benchmark families only solvable due to enumeration
- Overall the enumerative strategies lead to a virtual portfolio of CVC4 solving 712 more problems

Comparison against other instantiation-based SMT solvers



- Portfolios run without interleaving strategies (not supported by Z3)
- Z3 uses several optimizations for **e** not implemented in CVC4
- Z3 does not implement **c**

Restricting Enumeration Space

- Strengthened Herbrand Theorem allows restriction to $\mathbf{T}(\mathsf{E})$
- *Sort inference* [CS03] reduces instantiation space by computing more precise sort information
  - $\mathsf{E} \cup \mathsf{Q} = \{a \not\simeq b,\ f(a) \simeq c\} \cup \{P(f(x))\}$
    - $a, b, c, x : \tau$
    - $f : \tau \to \tau$ and $P : \tau \to$ Bool.
  - This is equivalent to $\mathsf{E}^s \cup \mathsf{Q}^s = \{a_1 \not\simeq b_1,\ f_{12}(a_1) \simeq c_2\} \cup \{P_2(f_{12}(x_1))\}$
    - $a_1, b_1, x_1 : \tau_1$
    - $c_2 : \tau_2$
    - $f_{12} : \tau_1 \to \tau_2$ and $P : \tau_2 \to$ Bool
  - **u** would derive e.g. $\{x \mapsto c\}$ for $\mathsf{E} \cup \mathsf{Q}$, while for $\mathsf{E}^s \cup \mathsf{Q}^s$ the instantiation $\{x_1 \mapsto c_2\}$ is not well-sorted.

Term Ordering

Instantiations are enumerated according to the order

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

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

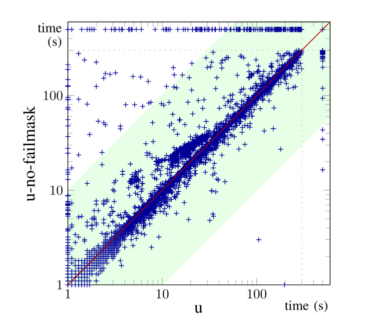
If $a \prec b \prec c$, then

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

- We consider instantiations with c only after considering all cases with a and b
- Goal is to introduce new terms less often
- Order on $\mathbf{T}(\mathrm{E})$ fixed for finite set of terms $t_1 \prec \ldots \prec t_n$
  - Instantiate in order with $t_1, \ldots, t_n$
  - Then choose new non-congruent term $t \in \mathbf{T}(\mathrm{E})$ and have $t_n \prec t$

Fair and Adventurous Enumeration of Quantifier Instantiations [JBFR21]

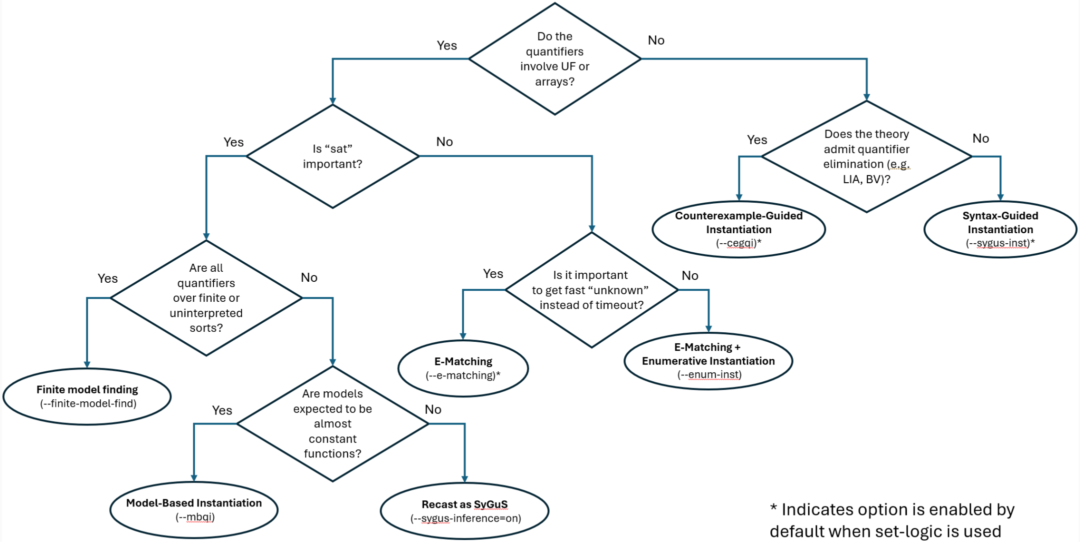
- Recently we tried a number of different orderings for tuples
  - Significant orthogonality, no clear winner
- In the process we improved the generation of instances in $\mathbf{u}$ to remove redundant tuples
  - A subset of the tuple is enough to lead to an entailed instance
  - A subset of the tuple is enough to lead an equivalent, modulo rewriting, instance



- UF, UFLIA, and UFNIA, totaling 31314 problems

So what quantifier handling techniques do I use if I have a quantified SMT problem?

A flowchat to pick quantifier handling strategies, at least in cvc5
(courtesy of Andy Reynolds)



* Indicates option is enabled by default when set-logic is used

Conclusion

- SMT solvers mainly use instantiation techniques to handle quantified formulas
- They are often incomplete, slow, or fragile
- But they also enable many successful use cases in e.g. software verification and automation for proof assistants
- There are many options, ask developers.

An introduction to SMT solving with quantifiers

Haniel Barbosa, Universidade Federal de Minas Gerais

SAT/SMT/AR Summer School

2024–06–27, LORIA–Inria, Université de Lorraine, Nancy, FR

CCFV calculus

$$\frac{E_\sigma \Vdash_E x \not\simeq y \wedge C}{E_\sigma \Vdash_E \bigvee_{[t],\,[t']\in E^{CC},\, E\models t\not\simeq t'} (x \simeq t \wedge y \simeq t' \wedge C)}\ \textsc{Dvar}$$

$$\frac{E_\sigma \Vdash_E x \not\simeq f(\bar{s}) \wedge C}{E_\sigma \quad \Vdash_E \quad \bigvee_{\substack{[t],\,[t']\in E^{CC},\\ E\models t\not\simeq t',\, f(\bar{t'})\in[t']}} (x \simeq t \wedge s_1 \simeq t'_1 \wedge \cdots \wedge s_n \simeq t'_n \wedge C)}\ \textsc{Dfapp}$$

$$\frac{E_\sigma \Vdash_E f(\bar{u}) \not\simeq g(\bar{s}_m) \wedge C}{E_\sigma \quad \Vdash_E \quad \bigvee_{\substack{[t],\,[t']\in E^{CC},\, E\models t\not\simeq t',\\ f(\bar{t})\in[t],\, g(\bar{t'}_m)\in[t']}} \begin{pmatrix} u_1 \simeq t_1 \wedge \cdots \wedge u_n \simeq t_n \ \wedge \\ s_1 \simeq t'_1 \wedge \cdots \wedge s_m \simeq t'_m \wedge C \end{pmatrix}}\ \textsc{Dgen}$$

$$\frac{E_\sigma \Vdash_E C_1 \vee C_2}{E_\sigma \Vdash_E C_1 \qquad E_\sigma \Vdash_E C_2}\ \textsc{Split} \qquad \frac{E_\sigma \Vdash_E \ell \wedge C}{E_\sigma \Vdash_E C}\ \textsc{Yield} \quad \text{if } E \models \ell$$

$$\frac{E_\sigma \Vdash_E \ell \wedge C}{E_\sigma \Vdash_E \bot}\ \textsc{Fail} \quad \text{if } \ell \text{ is ground and } E \not\models \ell$$

References

[ABJ+13] Rajeev Alur, Rastislav Bodík, Garvit Juniwal, et al. "Syntax-guided synthesis". In: Formal Methods In Computer-Aided Design (FMCAD). IEEE, 2013, pp. 1–8.

[Bar16] Haniel Barbosa. "Efficient Instantiation Techniques in SMT (Work In Progress)". In: Practical Aspects of Automated Reasoning (PAAR). Ed. by Pascal Fontaine, Stephan Schulz, and Josef Urban. Vol. 1635. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 1–10.

[BFR17] Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. "Congruence Closure with Free Variables". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part II. Ed. by Axel Legay and Tiziana Margaria. Vol. 10206. Lecture Notes in Computer Science. 2017, pp. 214–230.

[BJ15] Nikolaj Bjørner and Mikolas Janota. "Playing with Quantified Satisfaction". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Ansgar Fehnker, Annabelle McIver, Geoff Sutcliffe, et al. Vol. 35. EPiC Series in Computing. EasyChair, 2015, pp. 15–27.

[Bjø10] Nikolaj Bjørner. "Linear Quantifier Elimination as an Abstract Decision Procedure". In: International Joint Conference on Automated Reasoning (IJCAR). Ed. by Jürgen Giesl and Reiner Hähnle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 316–330.

References ii

[BKPU16] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, et al. “Hammering towards QED”. In: J. Formalized Reasoning 9.1 (2016), pp. 101–148.

[BR15] Peter Backeman and Philipp Rümmer. “Theorem Proving with Bounded Rigid E-Unification”. In: Proc. Conference on Automated Deduction (CADE). Ed. by Amy Felty and Aart Middeldorp. Vol. 9195. Lecture Notes in Computer Science. Springer, 2015.

[CS03] Koen Claessen and Niklas Sörensson. “New Techniques that Improve MACE-style Finite Model Finding”. In: Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications. 2003.

[DCKP13] Claire Dross, Sylvain Conchon, Johannes Kanig, et al. “Adding Decision Procedures to SMT Solvers using Axioms with Triggers”. 2013.

[DNS05] David Detlefs, Greg Nelson, and James B. Saxe. “Simplify: A Theorem Prover for Program Checking”. In: J. ACM 52.3 (2005), pp. 365–473.

[GM09] 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.

[JBFR21] Mikolás Janota, Haniel Barbosa, Pascal Fontaine, et al. “Fair and Adventurous Enumeration of Quantifier Instantiations”. In: Formal Methods In Computer-Aided Design (FMCAD). IEEE, 2021, pp. 256–260.

[Kan63] Stig Kanger. "A Simplified Proof Method for Elementary Logic". In: Computer Programming and Formal Systems. Ed. by P. Braffort and D. Hirschberg. Vol. 35. Studies in Logic and the Foundations of Mathematics. Elsevier, 1963, pp. 87 –94.

[KV13] 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.

[Lei10] K. Rustan M. Leino. "Dafny: An Automatic Program Verifier for Functional Correctness". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Edmund M. Clarke and Andrei Voronkov. Vol. 6355. Lecture Notes in Computer Science. Springer, 2010, pp. 348–370.

[LHC+23] Andrea Lattuada, Travis Hance, Chanhee Cho, et al. "Verus: Verifying Rust Programs using Linear Ghost Types". In: Proc. ACM Program. Lang. 7.OOPSLA1 (2023), pp. 286–315.

[LP16] K. Rustan M. Leino and Clément Pit-Claudel. "Trigger Selection Strategies to Stabilize Program Verifiers". In: Computer Aided Verification (CAV). Ed. by Swarat Chaudhuri and Azadeh Farzan. Vol. 9779. Lecture Notes in Computer Science. Springer, 2016, pp. 361–381.

[MB07] 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.

[Mon10] David Monniaux. “Quantifier Elimination by Lazy Model Enumeration”. In: Computer Aided Verification (CAV). Ed. by Tayssir Touili, Byron Cook, and Paul B. Jackson. Vol. 6174. Lecture Notes in Computer Science. Springer, 2010, pp. 585–599.

[NPR+21] Aina Niemetz, Mathias Preiner, Andrew Reynolds, et al. “Syntax-Guided Quantifier Instantiation”. In: Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part II. Ed. by Jan Friso Groote and Kim Guldstrand Larsen. Vol. 12652. Lecture Notes in Computer Science. Springer, 2021, pp. 145–163.

[Ott08] Jens Otten. “leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic (System Descriptions)”. English. In: Automated Reasoning. Ed. by Alessandro Armando, Peter Baumgartner, and Gilles Dowek. Vol. 5195. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 283–291.

[RBF18] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. “Revisiting Enumerative Instantiation”. In: Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part II. Ed. by Dirk Beyer and Marieke Huisman. Vol. 10806. Lecture Notes in Computer Science. Springer, 2018, pp. 112–131.

[RBN+19] Andrew Reynolds, Haniel Barbosa, Andres Nötzli, et al. “cvc4sy: Smart and Fast Term Enumeration for Syntax-Guided Synthesis”. In: Computer Aided Verification (CAV), Part II. Ed. by Isil Dillig and Serdar Tasiran. Vol. 11562. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 74–83.

[RDK+15] Andrew Reynolds, Morgan Deters, Viktor Kuncak, et al. “Counterexample-Guided Quantifier Instantiation for Synthesis in SMT”. In: Computer Aided Verification (CAV). Ed. by Daniel Kroening and Corina S. Pasareanu. Vol. 9207. Lecture Notes in Computer Science. Springer, 2015, pp. 198–216.

References v

[RTG+13] Andrew Reynolds, Cesare Tinelli, Amit Goel, et al. "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.

[RTM14] 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.

[SCV19] Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. "Faster, Higher, Stronger: E 2.3". In: Proc. Conference on Automated Deduction (CADE). Ed. by Pascal Fontaine. Vol. 11716. Lecture Notes in Computer Science. Springer, 2019, pp. 495–507.