Towards higher-order unification in HOSMT

Haniel Barbosa
Andrew Reynolds
Cesare Tinelli

The University of Iowa

Daniel El Ouraoui
Pascal Fontaine

Inria informatics mathematics

ICMS 2018

2018-07-27, Notre Dame, IN, USA

Why higher-order logic?

Higher-Order logic

- Expressive
  - Mathematics
  - Verification conditions
- The language of proof assistants
  - Isabelle, Coq, Lean

Automation

- Hard to automatize
- Few provers to reason on it
  LEO-II, Leo-III, Satalax

Challenge

- New techniques for SMT
- Avoid automatic translation

Outline

- What we mean by higher-order logic
- A pragmatic extension of an SMT solver to higher-order
  - Ground decision procedure
  - Quantifiers and lambdas
  - Evaluation
- Towards higher-order unification

Fragments of interest

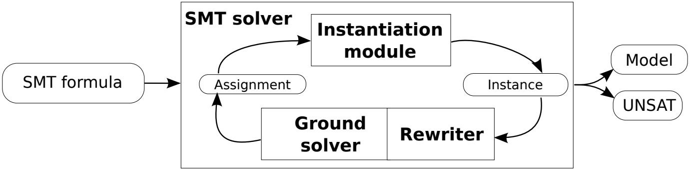
Features	Predicate calculus	λ-free	λ-calculus
function	✓	✓	✓
functional arguments	✗	✓	✓
quantification on objects	✓	✓	✓
quantification on functions	✗	✓	✓
partial applications	✗	✓	✓
anonymous functions	✗	✗	✓

- Henkin semantics
  - Function interpretations restricted to expressible terms
- Extensionality

$$\forall \bar{x}.\ f(\bar{x}) \simeq g(\bar{x}) \leftrightarrow f \simeq g$$

A pragmatic extension into HOSMT

A CDCL(T) SMT solver



- ▷ Rewriter simplifies terms

  $x + 0 \to x \qquad a \not\simeq a \to \bot \qquad (\mathrm{str.replace}\ x\ (\mathrm{str.{+}{+}}\ x\ x)\ y) \to x$

- ▷ Ground solver enumerates assignments E ∪ 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 $\{\forall xyz.\ f(x) \not\simeq f(z) \lor g(y) \simeq h(z)\}$

- ▷ Instantiation module generates instances of Q $f(a) \not\simeq f(b) \lor g(a) \simeq h(b)$

Our pragmatic extension

- λ-lifting at rewriter

$$\lambda x.\, t \rightarrow \forall x.\, f\, x \simeq t, \text{ where } f \text{ is a fresh symbol}$$

- Explicit applications introduced during solving
  - Lazy encoding
  - Lazy extensionality lemmas
  - Polynomial model construction for partial functions

- Extending E-matching algorithm for instantiation

Handling partial applications: applicative encoding

encoding

For all terms of the shape $(((f_{\tau_1 \to \ldots \to \tau_n \to \sigma}\; a_1) \ldots)\; a_n)) : \sigma$ given a family of symbols @ we have the translation App defined as following:

$$\mathsf{App}(((f\; a_1) \ldots)\; a_n)) = @(@(\ldots @(f, a_1), \ldots, a_n))$$

$f\,a\,b \simeq b \wedge f\,a\,(f\,a\,b) \simeq g\,b$

$@(@(f,a),b) \simeq b \wedge @(@(f,a),\, @(@(f,a),b)) \simeq @(g,b)$

where f, g become constant symbols

`app translation`

Lazy encoding

- Turn all partial applications into total
- Use first-order procedure on $\mathsf{App}(E)$
- Add remaining equalites between regular terms
  $E' = \mathsf{App}(E) \cup \{\mathsf{App}(f(a_1, ..., a_n)) \simeq f(a_1, ..., a_n),\ ...\}$
- **Only for partial function applications!**
- Check again E'

Example

$f\ a \simeq g \wedge f(a, a) \not\simeq g(a) \wedge g(a) \simeq h(a) \Rightarrow\ \{@(f, a) \simeq g,\ f(a, a) \not\simeq g(a),\ g(a) \simeq h(a)\} \subseteq E$

Lazy encoding

- Turn all partial applications into total
- Use first-order procedure on $\mathsf{App}(E)$
- Add remaining equalites between regular terms
  $E' = \mathsf{App}(E) \cup \{\mathsf{App}(f(a_1, ..., a_n)) \simeq f(a_1, ..., a_n),\ ...\}$
- **Only for partial function applications!**
- Check again E'

Example

$$f\ a \simeq g \wedge f(a, a) \not\simeq g(a) \wedge g(a) \simeq h(a) \Rightarrow\ \{@(f, a) \simeq g,\ f(a, a) \not\simeq g(a),\ g(a) \simeq h(a)\} \subseteq E$$

$$E \cup \{@(@(f, a), a) \simeq f(a, a),\ @(g, a) \simeq g(a)\} \Rightarrow @(@(f, a), a) \simeq @(g, a)$$

Lazy encoding

- Turn all partial applications into total
- Use first-order procedure on $\mathsf{App}(E)$
- Add remaining equalites between regular terms
  $E' = \mathsf{App}(E) \cup \{\mathsf{App}(f(a_1, ..., a_n)) \simeq f(a_1, ..., a_n), \ ...\}$
- **Only for partial function applications!**
- Check again E'

Example

$$f\ a \simeq g \wedge f(a,a) \not\simeq g(a) \wedge g(a) \simeq h(a) \Rightarrow \ \{@(f,a) \simeq g,\ f(a,a) \not\simeq g(a),\ g(a) \simeq h(a)\} \subseteq E$$

$$E \cup \{@(@(f,a),a) \simeq f(a,a),\ @(g,a) \simeq g(a)\} \Rightarrow @(@(f,a),a) \simeq @(g,a)$$

Handling extentionality

$$(\forall \bar{x}\ f(\bar{x}) \simeq g(\bar{x})) \leftrightarrow f \simeq g$$

- ▷ The "$\leftarrow$" direction is ensured by the functional congruence axiom:
$$f \simeq g \rightarrow (\forall \bar{x}\ f(\bar{x}) \simeq g(\bar{x}))$$

- ▷ The "$\rightarrow$" direction is ensured by $f(\bar{k}) \not\simeq g(\bar{k})$ for some Skolem $\bar{k}$

- ▷ $f(\bar{k}) \not\simeq g(\bar{k}) \vee f \simeq g$ is added for each pair of functions of finite type

Avoiding exponential model construction

Functions are interpreted as if-then-else:

$$M(f) = \lambda x\, \mathsf{ite}(x \simeq t_1, s_1, \ldots \mathsf{ite}(x \simeq t_{n-1}, s_{n-1}, s_n) \ldots)$$

Partial applications can lead to exponentially many cases!

$$\begin{array}{l} f_1(0) \simeq f_1(1) \wedge f_1(1) \simeq f_2 \\ f_2(0) \simeq f_2(1) \wedge f_2(1) \simeq f_3 \\ f_3(0) \simeq f_3(1) \wedge f_3(1) \simeq 2 \end{array}$$

8 ite entries to model that $f_1(x, y, z) \simeq 2$, for $x, y, z \in \{0, 1\}$

Polynomial construction in the "depth" of functions chain

$$\begin{array}{ll} M(f_1) = \lambda xyz. & \mathsf{ite}(x \simeq 0, M(f_2)(y,z), \mathsf{ite}(x \simeq 1, M(f_2)(y,z), \_)) \\ M(f_2) = \lambda xy. & \mathsf{ite}(x \simeq 0, M(f_3)(y), \mathsf{ite}(x \simeq 1, M(f_3)(y), \_)) \\ M(f_3) = \lambda x. & \mathsf{ite}(x \simeq 0, 2, \mathsf{ite}(x \simeq 1, 2, \_)) \end{array}$$

Instantiation strategies: *trigger-based*

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

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

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

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

λ-free E-matching for HO trigger-based instantatiation

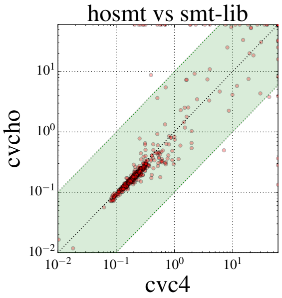
- Applicative encoding allows lifting of FO E-matching

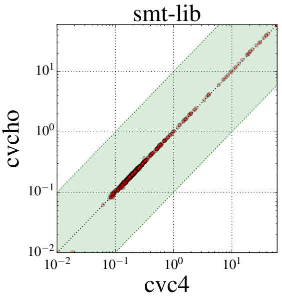
- Dedicate indexing techniques to account for equality of functions
  - In FO term indexing is done by head of applications

  - In HO two applications can be equals with different head symbol

$$@(f, a) \simeq g \models (g(x) \simeq f(a, b))\{x \mapsto b\}$$

Evaluation





	hosmt		*smt-lib*	
	#unsat	avg time (s)	#unsat	avg time (s)
CVC4-HO	648	1.08	662	1.02
CVC4	4	0.06	662	1.01

CVC4 configurations on "Judgement day" benchmarks with 60s timeout.

Towards higher-Order unification

Current limitations

$$\varphi = q(k(0,1)) \wedge \neg p(k(0,0)) \wedge \forall f y z.\, p(f(y,z)) \vee \neg q(f(1,y))$$

- UNSAT requires e.g. instantiation $\{f \mapsto \lambda w_1 w_2.\, k(0, w_1),\ y \mapsto 0,\ z \mapsto 0\}$
- Huet's algorithm for HO unification can find such instantiations
- HO unification is undecidable!

A first extension

- Use first-order matches to generate higher-order matches

- Enumerate permutations of arguments

$$\langle f(y,z),\, k(0,0)\rangle \Rightarrow \{f \mapsto k,\; y \mapsto 0,\; z \mapsto 0\}$$

$$f \quad \mapsto \lambda w_1 w_2.\, k(w_1, w_2)$$

A first extension

- Use first-order matches to generate higher-order matches

- Enumerate permutations of arguments

$$\langle f(y,z),\, k(0,0)\rangle \Rightarrow \{f \mapsto k,\; y \mapsto 0,\; z \mapsto 0\}$$

$$\begin{aligned} f &\quad \mapsto \lambda w_1 w_2.\, k(w_1, w_2) \\ f &\quad \mapsto \lambda w_1 w_2.\, k(w_2, w_1) \end{aligned}$$

A first extension

- Use first-order matches to generate higher-order matches
- Enumerate permutations of arguments

$$\langle f(y, z),\ k(0, 0)\rangle \Rightarrow \{f \mapsto k,\ y \mapsto 0,\ z \mapsto 0\}$$

$$\begin{aligned} f &\quad \mapsto \lambda w_1 w_2.\, k(w_1, w_2) \\ f &\quad \mapsto \lambda w_1 w_2.\, k(w_2, w_1) \\ f &\quad \mapsto \lambda w_1 w_2.\, k(0, w_1) \end{aligned}$$

A first extension

- Use first-order matches to generate higher-order matches

- Enumerate permutations of arguments

$$\langle f(y,z),\, k(0,0)\rangle \Rightarrow \{f \mapsto k,\, y \mapsto 0,\, z \mapsto 0\}$$

$$\begin{array}{ll} f & \mapsto \lambda w_1 w_2.\, k(w_1, w_2) \\ f & \mapsto \lambda w_1 w_2.\, k(w_2, w_1) \\ f & \mapsto \lambda w_1 w_2.\, k(0, w_1) \end{array}$$

A first extension

- Use first-order matches to generate higher-order matches
- Enumerate permutations of arguments

$$\langle f(y,z),\, k(0,0)\rangle \Rightarrow \{f \mapsto k,\; y \mapsto 0,\; z \mapsto 0\}$$

$$\begin{aligned}
f &\mapsto \lambda w_1 w_2.\, k(w_1, w_2) \\
f &\mapsto \lambda w_1 w_2.\, k(w_2, w_1) \\
f &\mapsto \lambda w_1 w_2.\, k(0, w_1) \\
f &\mapsto \lambda w_1 w_2.\, k(w_1, 0)
\end{aligned}$$

A first extension

- Use first-order matches to generate higher-order matches

- Enumerate permutations of arguments

$$\langle f(y,z),\, k(0,0)\rangle \Rightarrow \{f \mapsto k,\ y \mapsto 0,\ z \mapsto 0\}$$

$$\begin{array}{ll} f & \mapsto \lambda w_1 w_2.\, k(w_1, w_2) \\ f & \mapsto \lambda w_1 w_2.\, k(w_2, w_1) \\ f & \mapsto \lambda w_1 w_2.\, k(0, w_1) \\ f & \mapsto \lambda w_1 w_2.\, k(w_1, 0) \\ f & \mapsto \lambda w_1 w_2.\, k(0, w_2) \end{array}$$

A first extension

- Use first-order matches to generate higher-order matches

- Enumerate permutations of arguments

$$\langle f(y,z),\, k(0,0)\rangle \Rightarrow \{f \mapsto k,\; y \mapsto 0,\; z \mapsto 0\}$$

$$\begin{aligned}
f &\mapsto \lambda w_1 w_2.\, k(w_1, w_2)\\
f &\mapsto \lambda w_1 w_2.\, k(w_2, w_1)\\
f &\mapsto \lambda w_1 w_2.\, k(0, w_1)\\
f &\mapsto \lambda w_1 w_2.\, k(w_1, 0)\\
f &\mapsto \lambda w_1 w_2.\, k(0, w_2)\\
f &\mapsto \lambda w_1 w_2.\, k(w_2, 0)
\end{aligned}$$

A first extension

- Use first-order matches to generate higher-order matches

- Enumerate permutations of arguments

$$\langle f(y,z),\, k(0,0)\rangle \Rightarrow \{f \mapsto k,\ y \mapsto 0,\ z \mapsto 0\}$$

$$\begin{aligned}
f &\mapsto \lambda w_1 w_2.\, k(w_1, w_2) \\
f &\mapsto \lambda w_1 w_2.\, k(w_2, w_1) \\
f &\mapsto \lambda w_1 w_2.\, k(0, w_1) \\
f &\mapsto \lambda w_1 w_2.\, k(w_1, 0) \\
f &\mapsto \lambda w_1 w_2.\, k(0, w_2) \\
f &\mapsto \lambda w_1 w_2.\, k(w_2, 0) \\
f &\mapsto \lambda w_1 w_2.\, k(0, 0)
\end{aligned}$$

Lifting CCFV framework to HO unification

- Congruence Closure with Free Variables is a framework for E-unification in SMT [Barbosa et al. TACAS'17]
- Basis for conflict-based instantiation [Reynolds et al. FMCAD'14]
- We will incorporate into the framework the rules for HO matching and HO unification
- Future implementation in CVC4 and VERIT

Conclusions

- Presented a pragmatic extension of an SMT solver to HOSMT
- On par with encoding-based approach
  - even without further optimizations!
- Towards effective and refutationally complete calculus
  - Extend CCFV framework
    - Pattern unification
    - (Bounded) higher-order unification
  - Combination with function synthesis approaches
- Other challenges: inductive reasoning, ...