Towards
Fast Interactive Verification
through
Strong Higher-Order Automation

Jasmin Blanchette
Pascal Fontaine
Stephan Schulz
Uwe Waldmann
Vision: Take the Hard Labor out of Interactive Verification

Push button automation for proof assistants (e.g. Coq) based on efficient higher-order (HO) provers
APPLICATION: A VERIFIED “EASYCHAIR”

“PC members cannot review papers if they have a conflict of interest”

Proof today:

using assms proof induction
  case (Step s a) thus ?case
  proof (cases a)
    case (Cact ca) show ?thesis
      using Step pref_Conflict_isRev reach.Step by simp
    next
    case (Uact ua) show ?thesis
      proof (cases ua)
        case (uPref confID uID p paperID pref)
          thus ?thesis using Step unfolding Uact uPref isRev_def2
            by (blast dest: pref_Conflict_isRevNth reach.Step)
        qed (insert Step,
          simp add: Uact isRev_def2 u_defs pref_Conflict_isRevNth_def)+
      qed simp+
    next
    case (UUact uua) show ?thesis using Step unfolding UUact isRev_def2
      by (meson IO_Automaton.reach.Step pref_Conflict_isRevNth)
    qed simp+
  qed (simp add: istate_def)

Induction Rule
Simplifier
Arithmetic Procedure
General Reasoner
First-Order Provers via SLEDGEHAMMER

fully automatic
Proof today:

```plaintext
using assms
proof (induction case (Step s a) thus ?case
proof (cases a)
  case (Cact ca) show ?thesis
    using Step pref_Conflict_isRev reach.Step by simp
next
  case (Uact ua) show ?thesis
    proof (cases ua)
      case (uPref confID uID p paperID pref)
      thus ?thesis using Step unfolding Uact uPref isRev_def2
        by (blast dest: pref_Conflict_isRevNth reach.Step)
      qed (insert Step,
        simp add: Uact isRev_def2 u_defs pref_Conflict_isRevNth_def)+
    next
  case (UUact uua) show ?thesis using Step unfolding UUact isRev_def2
    by (meson IO_Automaton.reach.Step pref_Conflict_isRevNth)
  qed simp+
  qed (simp add: istate_def)
```
**Application: A Verified “EasyChair”**

“PC members cannot review papers if they have a conflict of interest”

**Proof after Matryoshka:**

```
using assms proof induction
  case (Step s a) thus ?case
  proof (cases a)
    case (Cact ca) show ?thesis
      using Step pref_Conflict_isRev reach.Step by simp
    next
    case (Uact ua) show ?thesis
      proof (cases ua)
        case (uPref confID uID p paperID pref)
        thus ?thesis using Step unfolding Uact uPref isRev_def2
          by (blast dest: pref_Conflict_isRevNth reach.Step)
      qed (insert Step,
        simp add: Uact isRev_def2 u_defs pref_Conflict_isRevNth_def)+
    next
    case (UUact uua) show ?thesis using Step unfolding UUact isRev_def2
      by (meson IO_Automaton.reach.Step pref_Conflict_isRevNth)
    qed simp+
  qed (simp add: istate_def)
```
Our Grand Challenge

Create **efficient proof calculi** and **higher-order provers** targeting proof assistants and their applications to software and hardware development

- by **fusing and extending** two lines of research: automatic proving & interactive proving

Scientific Objectives

- **SO1.** Extend superposition and SMT to higher-order logic
- **SO2.** Design practical **methods and heuristics** based on benchmarks
- **SO3.** Conceive **stratified architectures** to build higher-order provers
- **SO4.** Integrate our provers into proof assistants (Isabelle, Lean, TLA+)
SO1—Higher-Order Superposition ($\lambda$SUP)

First-order rule:

\[
\frac{D' \lor t \approx t' \quad C' \lor s[u] \not\approx s'}{(D' \lor C' \lor s[t'] \not\approx s')\sigma}
\]

where \( \sigma = \text{mgu}(t, u) \) \( u \) is not a variable \( t\sigma \not\approx t'\sigma \) \( s\sigma \not\approx s'\sigma \)

\( (t \approx t')\sigma \) is strictly maximal in \( (D' \lor t \approx t')\sigma \) and no selection \( (s \not\approx s')\sigma \) is maximal in \( (C' \lor s \not\approx s')\sigma \) or selected
**SO1—Higher-Order Superposition (λSUP)**

First-order rule:

\[
\begin{align*}
D' \lor t &\approx t' & C' \lor s[u] &\not\equiv s' \\
\hline
(D' \lor C' \lor s[t'] \not\equiv s')\sigma
\end{align*}
\]

where \( \sigma = \text{mgu}(t, u) \), \( u \) is not a variable, \( t\sigma \not\equiv t'\sigma \), \( s\sigma \not\equiv s'\sigma \),

- \((t \approx t')\sigma\) is strictly maximal in \((D' \lor t \approx t')\sigma\) and no selection
- \((s \not\equiv s')\sigma\) is maximal in \((C' \lor s \not\equiv s')\sigma\) or selected

- We need **sequences of unifiers**
**SO1—Higher-Order Superposition ($\lambda$SUP)**

First-order rule:

\[
\frac{D' \lor t \equiv t' \quad C' \lor s[u] \not\equiv s'}{(D' \lor C' \lor s[t'] \not\equiv s')\sigma} \quad \text{SUP-LEFT}
\]

where \( \sigma = \text{mgu}(t, u) \) \( u \) is not a variable \( t\sigma \not\preceq t'\sigma \) \( s\sigma \not\preceq s'\sigma \)

\( (t \equiv t')\sigma \) is strictly maximal in \( (D' \lor t \equiv t')\sigma \) and no selection

\( (s \not\equiv s')\sigma \) is maximal in \( (C' \lor s \not\equiv s')\sigma \) or selected

- We need sequences of unifiers
- We need higher-order term ordering
- We also want proof-assistant-style HO rewriting
**SO3—Stratified Architecture**

Inspired by Nelson–Oppen (SMT)

Base FO provers: E & veriT

Some scientific challenges:

- How to exploit derived FO formulas and/or candidate models to guide HO quantifier instantiation?
- How to generate certificates for reconstruction in proof assistants?

First-Order Prover (e.g. veriT)

Matryoshka Prover (e.g. veriHOT)
SO4—Connection with Proof Assistants

Dependent Type Theory

Classical Higher-Order Logic

Set Theory

Lean

Agda
Coq
Matita

Isabelle/HOL

HOL4
HOL Light
PVS

TLA+

Isabelle/ZF
Mizar
Rodin (Event-B)
# The matryoshka Team

**Scientific Leader:** Jasmin Blanchette  
**Senior Collaborator:** Pascal Fontaine  
**Postdoctoral Researchers:** Johannes Hölzl, Rob Lewis  
**Ph.D. Students:** Alex Bentkamp, Daniel El Ouraoui, Hans-Jörg Schurr, Petar Vukmirović  
**Associated Members:** Stephan Schulz, Uwe Waldmann  
**Other Collaborators:** Haniel Barbosa, Simon Cruanes, Simon Robillard, & more

<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Initials</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jasmin Blanchette</td>
<td></td>
<td>Adam</td>
</tr>
<tr>
<td>Pascal Fontaine</td>
<td></td>
<td>Ncy</td>
</tr>
<tr>
<td>Johannes Hölzl</td>
<td></td>
<td>Adam</td>
</tr>
<tr>
<td>Rob Lewis</td>
<td></td>
<td>Adam</td>
</tr>
<tr>
<td>Alex Bentkamp</td>
<td></td>
<td>Adam</td>
</tr>
<tr>
<td>Daniel El Ouraoui</td>
<td></td>
<td>Ncy</td>
</tr>
<tr>
<td>Hans-Jörg Schurr</td>
<td></td>
<td>Ncy</td>
</tr>
<tr>
<td>Petar Vukmirović</td>
<td></td>
<td>Adam</td>
</tr>
<tr>
<td>Stephan Schulz</td>
<td></td>
<td>Stgt</td>
</tr>
<tr>
<td>Uwe Waldmann</td>
<td></td>
<td>SB</td>
</tr>
<tr>
<td>Haniel Barbosa</td>
<td></td>
<td>Ncy</td>
</tr>
<tr>
<td>Simon Cruanes</td>
<td></td>
<td>Ncy</td>
</tr>
<tr>
<td>Simon Robillard</td>
<td></td>
<td>Gbg</td>
</tr>
<tr>
<td>&amp; more</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
http://matryoshka.gforge.inria.fr
A lot of work has gone into engineering the individual proof assistants. Maybe too little has been into developing compositional methods and tools with a **broad applicability** across systems?

Have we done enough for automated reasoning to be used as a tool, where it is needed, for real-life applications? Aren't we creating a FOL playground, whereas the **world expects HO**?