Sledgehammer: a Saga

Lawrence C Paulson

Vampire Workshop, 1 July 2024, Nancy, France
• A suggestion by Andrei Voronkov (at IJCAR 2001 in Siena?): let’s combine Isabelle with a **real** theorem prover

• Meetings with Weidenbach and Siekmann in Saarbrücken

• A grant of £249,905 starting in January 2004

• ... and a report of early results that July, at IJCAR 2004

• First release early in 2007, integrating Isabelle with E, SPASS, Vampire
Some precursors and influences

Ωmega (Siekmann, Benzmüller et al.)

KIV + $\_3T^A^P$ (Ahrendt, Beckert et al.)

Integrating Gandalf and HOL (Hurd)

Coq + Bliksem (Bezem et al.)

Unfortunately, they demanded too much work from the user
“Now that 2GHz processors are commonplace, we should abandon the traditional mode of interaction, where the proof tool does nothing until the user types a command. Background processes ... should try to prove the outstanding subgoals.”

[from the original proposal, 2003]
Original design criteria

- **Easy invocation** (1-click, or even 0-click)
- **Automatic translation** from higher-order logic to first-order logic
- Instant access to the *entire lemma library*, with relevance checking
- Result as a *proof certificate*
  - To avoid having to *rerun* the search
  - To avoid trusting external tools
First Working Prototype
The tasks

- Relevance filtering
- Translating to FOL: types
- Translating to FOL: \(\lambda\)-bindings
- Proof reconstruction
Relevance filtering

[AKA premise selection]

- An Isabelle session may have 10,000+ accessible facts
- Theorem provers (at that time) could cope with a couple of hundred
- Relevance may be more obvious to the interactive prover (cf KIV)
- We adopted a crude approach based on symbol occurrences
Translating to FOL: types

A fully typed translation is **heavy** (quadratic), burying the formulas themselves

$$E = mc^2$$

$$((( = )E)(\times m(\uparrow c2)))$$

So I adopted a **partially typed translation** (unsound!)

... handling **polymorphism** and **type classes**

[Joe Hurd had success with a **completely typeless** translation]
Translating to FOL: $\lambda$-bindings

- Translation approaches (*neither works well!*): include:
  1. **Combinator** form S, K, I, B, C, ...
  2. **$\lambda$-lifting** (generating new function definitions)

- Have an explicit “apply” function and “is true” predicate for booleans, but **full higher-order reasoning** is not possible

- All of this omitted if the problem is **fully first-order**: in fact a “smooth” translation is possible
Thousands of hours of testing
Here we compare various translations by % problems solved

- constant (FO)
- constant
- partial
- full (FO)
- full

Percent solved vs. Runtime per problem (seconds)
Proof reconstruction

Proofs given by ATPs are too ambiguous to use

So we decided to use ATPs as powerful **relevance filters**

From the proof we extract nothing but the fact names

... giving them to one of Isabelle's own proof tools

Hurd's **metis**, a superposition prover integrated with the kernel
Working by February 2007 isn’t hard, but requires four separate facts.
Also with single-step proofs

```
lemma "[\{b < a; 0 < c\} \implies -a * c < - (c * (b::int))\]"
```

```
IS08--**--XEmacs: Test.thy  (Isar script XS:isabelle/s PenDel Font Abbrev;
Subgoal 1: Success.
proof (neg_clausify)
  assume 0: "b < a"
  assume 1: "0 < c"
  assume 2: "\neg (- a * c < - (c * b))"
  have 3: "\forall X1 X2. Numeral.Min * X1 < - X2 \lor \neg X2 < X1"  
    by (metis neg_less_iff_less mult_Min)
  have 4: "\neg c * b < c * a"
    by (metis 3 zmult_commute zmult_assoc mult_Min 2)
  have 5: "\neg b < a"
    by (metis 4 zmult_zless_mono2 1)
  show "False"
    by (metis 5 0)

[b < a; 0 < c] \implies -a * c < - (c * b)
```
Others Take Over
Issues with the prototype

- **Unsound translations** (resulting in worthless "proofs")
- **Simplistic methods** (esp. relevance filtering)
- ** Truly horrible code**
The all-new sledgehammer

- A family of efficient, sophisticated and sound translations for monomorphic and polymorphic types
- An ML based relevance filter for premise selection
- Additional external provers, notably SMT solvers such as Z3
- ... justified by additional internal provers, including Isabelle's Z3

The work of Jasmin Blanchette, Sascha Böhme and Tobias Nipkow
Running three different theorem provers (E, SPASS and Vampire) each for **five seconds** solves as many problems as running the best theorem prover (Vampire) for **two full minutes**.
Higher-order superposition

- An effective alternative to translating λ-calculus into first-order logic
- A sound and complete calculus for higher-order logic with polymorphism, extensionality, Hilbert choice, and Henkin semantics
- And a term ordering to limit the search space
- And an implementation! Zipperposition outperforms all other higher-order theorem provers

The work of Bentkamp, Blanchette, Tourret, Vukmirovic
Giving back to the ATP community
(By verifying their theoretical canon)

A verified SAT solver framework with learn, forget, restart, and incrementality

A verified prover based on ordered resolution

Formalizing Bachmair and Ganzinger’s ordered resolution prover

Formalized superposition
Impact
Synergy with structured proofs

Every line justified by sledgehammer!
... hence, easier for beginners

• No more memorising lists of built-in facts
• No more learning obscure tactics for pushing symbols around
• The key skill: **thinking up intermediate goals**
• Given the proof structure, *Sledgehammer does the rest!*
Turning English into proofs using AI
Draft, sketch and prove: Jiang et al.

Statement
If $\text{gcd}(n, 4) = 1$ and $\text{lcm}(n, 4) = 28$, show that $n$ is 7.

Informal proof
We know that $\text{gcd}(a, b) \cdot \text{lcm}(a, b) = ab$, hence $1 \cdot 28 = n \cdot 4$.
Then $n = 1 \cdot 28/4 = 7$,
completing the proof. ■

Formal sketch
have cl: “$1 \cdot 28 = n \cdot 4$”
using assms
<proof>
then have c2: “$n = 1 \cdot 28/4$”
<proof>
then show ?thesis
<proof>

Verified formal proof
have cl: “$1 \cdot 28 = n \cdot 4$”
using assms
by (smt (z3) prod_gcd_lcm_nat)
then have c2: “$n = 1 \cdot 28/4$”
by auto
then show ?thesis
by auto

Diagram:
- Informal Proof Writer: Draft informal proof
- Autoformalizer: Generate formal sketch
- Off-the-shelf Prover: Prove remaining gaps
Strong growth in lines of code
Isabelle's Archive of Formal Proofs
New applications for ATPs themselves

A limitless supply of users with tough problems

Motivation for extensions such as types and polymorphism

Strong justification for automating higher-order logic, e.g. in CVC and E
And other hammers, notably for HOL, Coq and Lean (forthcoming)
Hopes for the future

• Strong support for problems involving $\lambda$-binding

• Genuine, powerful higher-order reasoning

• **Hints to users**, say about possibly missing assumptions or lemmas

• A truly effective and sound integration with AI
Acknowledgements

Claire Quigley: process management

Jia Meng: relevance, HO translations

Kong Woei Susanto: Metis

EPSRC Engineering and Physical Sciences Research Council

Project GR/S57198/01 Automation for Interactive Proof

Plus the group at TU Munich that created the second generation