Source-Level Proof Reconstruction for Interactive Proving

Lawrence C. Paulson and Kong Woei Susanto
Computer Laboratory, University of Cambridge

Motivation

- Interactive provers are good for specifying complex systems, but proving theorems requires too much work.
- Linking them to automatic provers can reduce the cost of using them.
- Trusting the output of a big system (including the linkup code) goes against the LCF tradition and is unsafe.
- Reconstruction lets us use techniques that are efficient but unsound.

Source-Level Proof Reconstruction

- The LCF architecture provides a kernel of inference rules, which is the basis of all proofs.
- Automatic tools may include a *proof reconstruction* phase, where they justify their reasoning to the proof kernel.

Why not instead deliver proofs in source form? Then users could *inspect* and *edit* them.

Isabelle Overview

- *Generic* proof assistant, supporting higher-order logic, ZF set theory, etc.
- *Axiomatic type classes* to express concepts such as *linear order* and *ring* through polymorphism.
- *Extensive lemma libraries*: real numbers (including non-standard analysis), number theory, hardware, …
- *Automation*: decision procedures, simplifier and prover, automatically referring to 2000 lemmas.

Automatic Provers

- *Resolution* is a general, powerful technique with full support for quantifiers and equations.
- The provers we use are Vampire, E and SPASS.
- Arithmetic is not built-in; however, Isabelle already provides support for the main decidable theories.
- Decision procedures have too narrow a focus. We seek automation that can be tried on *any* problem.

Overview of the Linkup

When the user invokes the "sledgehammer" command...

- The problem is Skolemized and converted to clause form, with higher-order features removed (all by inference).
- A simple relevance filter chooses a few hundred lemmas to include with the problem.
- Further clauses convey limited information about *types* and *type classes*.
- A resolution prover starts up (in the background).

Obstacles to Reconstruction with Automatic Provers

- *Ambiguities*: their output typically omits crucial information, such as which term is affected by rewriting.
- *Lack of standards*: automatic provers generate different output formats and employ a variety of inference systems.
- *Complexity*: a single automatic prover may use numerous inference rules with complicated behaviours.
- *Problem transformations*: ATPs re-order literals and make other changes to the clauses they are given.

Joe Hurd's Metis Prover

- Metis is a clean implementation of resolution, with an ML interface for LCF-style provers, originally HOL4.
- We provide *metis* as an Isabelle command, with internal proof reconstruction.
- We translate ATP output into a series of *metis* calls.
- Metis cannot replace leading provers such as Vampire, but it can usually *re-run* their proofs.

Porting Metis to Isabelle

- *Conversion to clauses*: use Isabelle's existing code for this task.
- *The 5 Metis inference rules*: implement using Isabelle's proof kernel.
- During type inference, recover type class information from the proof.
- *Ignore* clauses and literals that encode type classes.

Approaches to Proof Reconstruction via Metis

1. A *single call* to metis, with just the needed lemmas
   - The ATP merely serves as a *relevance filter*.
   - Parsing is trivial: we merely look for axiom numbers to see which lemmas were used.
2. A *line-by-line* reconstruction of the resolution proof
   - We translate the ATP proof into an ugly Isabelle proof.

Sutcliffe's TSTP Format

- **Thousands of Solutions from Theorem Provers**
- A standard for returning outcomes of ATP calls
- Proof lines have the form

cnf(<*name*>,<*formula_role*>,<*cnf_formula*><*annotations*>).

axiom, conjecture, etc.

referenced proof lines

A TSTP Axiom Line

- This line expresses the equation

$$X - X = 0$$

```
cnf(216,axiom,
    (c_minus(X,X,X3)=c_HOL_Ozero(X3) |
     ~class_OrderedGroup_Oab__group__add(X3)),
    file('BigO__bigo_bounded2_1', cls_right__minus__eq_1)).
```

A TSTP Conjecture Line

- This line expresses type information about the given problem. (The type variable `‘b` is in class `ordered_idom`.)
- Proof reconstruction must ignore it.

```
cnf(335,negated_conjecture,
    (class_Ring__and__Field_Oordered__idom(t_b)),
    file('BigO__bigo_bounded2_1', tfree_tcs)).
```

A TSTP Proof Step

- The E prover's inferences look like this.
- It conveys more information about the type variable `'b`, so it too must be ignored.

```
cnf(366,negated_conjecture,
    (class_OrderedGroup_Opordered__ab__group__add(t_b)),
    inference(spm,[status(thm)],
              [343,335,theory(equality)])).
```

What to Do with Various Proof Lines

- *Axiom reference*: delete, using instead the lemma name.
- *Type class inclusion*: delete entirely.
- *Conjecture clause*: copy it into the Isabelle proof, as an assumption.
- *Inference*: copy it into the Isabelle proof, justified by a call to *metis*.

Turning TSTP into Isabelle

- Parse TSTP format, recovering *proof structure*.
- Use type literals in clauses to recover *class constraints* on type variables.
- Use Isabelle's type inference to recover *terms*.
- Use Isabelle's pretty printer to generate *strings*.
- Combine strings to yield an *Isar structured proof*.

Collapsing of Proof Steps

We can shorten the proof by combining adjacent steps, giving *metis* more work to do!

- Some assertions aren't expressible in Isabelle: quantifications over types, type class inclusions.
- Some inferences are trivial (instantiating variables in another line) or become trivial once type literals are ignored.
- Some proofs are just intolerably long (a hundred lines).

A Typical Structured Proof

```
proof (neg_clausify)
fix x
assume 0: "⋀y. lb y ≤ f y"
assume 1: "¬ (0::'b) ≤ f x + - lb x"
have 2: "⋀X3. (0::'b) + X3 = X3"
  by (metis diff_eq_eq right_minus_eq)
have 3: "¬ (0::'b) ≤ f x - lb x"
  by (metis 1 compare_rls(1))
have 4: "¬ (0::'b) + lb x ≤ f x"
  by (metis 3 le_diff_eq)
show "False"
  by (metis 4 2 0)
qed
```

Future Ideas and Conclusions

- ATPs can help generate their own proof scripts!
- Scripts may need type annotations, which at present are highly repetitions.
- Redundant material, such as proofs of known facts, could be deleted.
- Can we produce scripts that look *natural*?

Acknowlegements

- Postdocs: Claire Quigley

- PhD student: Jia Meng

- Funding: EPSRC project GR/S57198/01 *Automation for Interactive Proof*

EPSRC Engineering and Physical Sciences Research Council