Cooperative Integration of an Interactive Proof Assistant and an Automated Prover

Adam Chlipala and George C. Necula
University of California, Berkeley
STRATEGIES 2006
We suggest a new idiom for semi-automated program verification. Implemented as a Coq tactic. In contrast to many automation tactics, it takes advantage of partial success through a possibility for cooperating interaction between human and automatic provers.

- Uses standard Nelson-Oppen prover features to structure the interaction.
ESC-Style Program Verification

∀ x, reach(mem, ls, x) ∧ x ≠ null → hd(mem, x) ≥ 0  When any hd(mem, v) is in the E-Graph
Instantiate with x := v

H1: ∀ x, reach(mem, ls, x) ∧ x ≠ null → hd(mem, x) ≥ 0
H2: ls ≠ null
H3: r ≥ 0
H4: h = ls.head
H5: h + r < 0
H6: reach(mem, ls, ls)
H7: reach(mem, ls, ls)

∀ m, ∀ x, reach(m, x, x)  Axiom:

When any n and v are in the E-Graph
Instantiate with m := n and x := v

∀ x, reach(mem, ls, x) ∧ x ≠ null → hd(mem, x) ≥ 0 ∧ ls ≠ null ∧ r ≥ 0 → hd(mem, ls) + r ≥ 0

(∀ x, reach(mem, ls, x) ∧ x ≠ null → hd(mem, x) ≥ 0) ∧ ls ≠ null ∧ r ≥ 0 → hd(mem, ls) + r ≥ 0
Let's Try Another....

```c
void splice(node* ls, node *mid, node *new) {
    mid->tail = new;
}
```

E-Graph

Prove False

Axiom:

\[
\forall m, \forall x, \forall y, reach(m, x, y) \rightarrow x = y \lor reach(m, sel(m, tl(x)), y)
\]

When any n, u, and v are in the E-Graph

When \( \text{any} \) \( \text{sel}(n, \text{tl}(u)) \) is in the E-Graph, \( y = v \)

Instantiate with \( m := n \) and \( x := v \)
ESC-Style Downsides

- Inductive proofs must **follow program structure**!
- If the decision procedure isn't smart enough, you're out of luck.
- Poor support for re-usable proof libraries
- Hard to use higher-order techniques

...but really convenient when it works!
Using Coq....

IH: \( \text{reach}(\text{mem}, \text{sel}(\text{mem}, \text{tl}(\text{ls})), \text{null}) \rightarrow \text{reach}(\text{upd}(\text{mem}, \text{tl}(\text{mid}), \text{new}), \text{sel}(\text{mem}, \text{tl}(\text{ls})), \text{null}) \)

H1: \( \text{reach}(\text{mem}, \text{null}, \text{null}) \)
H2: \( \text{reach}(\text{mem}, \text{sel}(\text{mem}, \text{tl}(\text{ls})), \text{null}) \)
H3: \( \text{reach}(\text{mem}, \text{new}, \text{null}) \)

But wait! How did Kettle prove that?
It would need to use a fact like:
\( \text{reach}(m, x, \text{null}) \land \text{reach}(m, v, \text{null}) \rightarrow \text{reach}(\text{upd}(m, u, v), x, \text{null}) \)

Kettle was unable to prove the goal.

Calls a Nelson-Oppen prover
The Initial Attempt

Kettle decided to do a case split on the equality of \( tl(ls) \) and new. It proved the case where they aren't equal to us.

Proof complete!

Since we can come up with a good instantiation heuristic for this lemma, we can add it to Kettle's knowledge base and have it used automatically next time....

Instantiate the lemma manually, and Kettle handles the rest!

Now we go prove the lemma we need as 
\[ \text{preserve\_reach} \].....

\begin{align*}
H4 &: tl(ls) = \text{new} \\
H5 &: \text{sel(upd(mem, tl(mid), new), tl(ls))} = \text{new} \\
H1 &: \text{reach(upd(mem, null)} \\
H2 &: \text{reach(mem)} \\
H3 &: \text{reach(mem, new, null)} \\
IH &: \text{reach(upd(mem, tl(mid), new, sel(mem, tl(ls)))} = \text{new} \\
H5 &: \text{sel(upd(mem, tl(mid), new), tl(ls))} = \text{new} \\
H1 &: \text{reach(mem, new)} \\
\end{align*}
An Even Better Way

- Run Kettle tactic to reduce goal into simpler subgoals.
- For each remaining subgoal $G$:
  - For each reachability hypothesis $H$:
    - Use elimination on $H$.
    - If Kettle can prove the subgoals completely, move on to next subgoal.
    - Otherwise, undo the elimination and try the next possible $H$.
  - If no suitable $H$ was found, leave $G$ for the user.

Ltac bounded_kettle' :=
  match goal with
  | [ H : reach _ _ _ | _ ] =>
    destruct H; kettle; fail
  end.
Ltac bounded_kettle :=
  kettle; try bounded_kettle'.
How It Works

Kettle

Case Splitting

Quantified Axioms with Triggers

Coq

Tactic

Ground Decision Procedures

Tactic
Proof Translation

∀ \( x, x = y \rightarrow P \rightarrow z = y \rightarrow Q \)
Reflective Proof Checking

“Do a case analysis on the equality of x and y. When \( x = y \), the result follows by arithmetic simplification. Otherwise, instantiate this lemma, and then...”

Definition \( \text{kettle\_prop} : \text{Set} := \ldots \)
Definition \( \text{kettle\_proof} : \text{kettle\_prop} \rightarrow \text{Set} := \ldots \)
Definition \( \text{interp\_prop} : \text{kettle\_prop} \rightarrow \text{Prop} := \ldots \)
Definition \( \text{check} : \forall (p : \text{kettle\_prop}) \ (pf : \text{kettle\_proof} p), \ \text{interp\_prop} p := \ldots \)

\( \lambda \)

Term in Dependent Type Theory

Proof

The goal is true because I have a Kettle proof of a proposition that \textit{compiles} to what you're looking for!
Implementation

- We've implemented this as a module linked into a custom Coq binary.
- Implementation tested in some case studies related to pointer-using programs
- In largest case study so far, our tactic helped reduce the number of proof script lines from 37 to 16.
  - ...and leads to less brittle proof scripts that adapt to small spec changes.
Conclusion

- Coq users benefit from a new way of automating parts of proofs.
- Historical users of ESC-style tools can use Coq as a more expressive way of driving their automated provers.
- Another contribution to the quest to find the sweet spot between expressivity and automation.