Cooperative Integration of an Interactive Proof Assistant and an Automated Prover

Adam Chlipala and George C. Necula
University of California, Berkeley
STRATEGIES 2006

Summary

- 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

$\forall x, reach(mem, ls, x) \wedge x \neq null \rightarrow hd(mem, x) \geq 0$

```
int sum(node* ls) {
    if (ls == null)
        return 0;
    else
        return ls->head
            + sum(ls->tail);
}
```

$result \geq 0$

Axiom:

$\forall m, \forall x, reach(m, x, x)$

When any *n* and *v* are in the E-Graph

Instantiate with *m* := *n* and *x* := *v*

When any *hd(mem, v)* is in the E-Graph

Instantiate with *x* := *v*

$H1: \forall x, reach(mem, ls, x) \wedge x \neq null \rightarrow hd(mem, x) \geq 0$

$H2: ls \neq null$

$H3: r \geq 0$

$H4: h = ls.head$

$H5: h + r < 0$

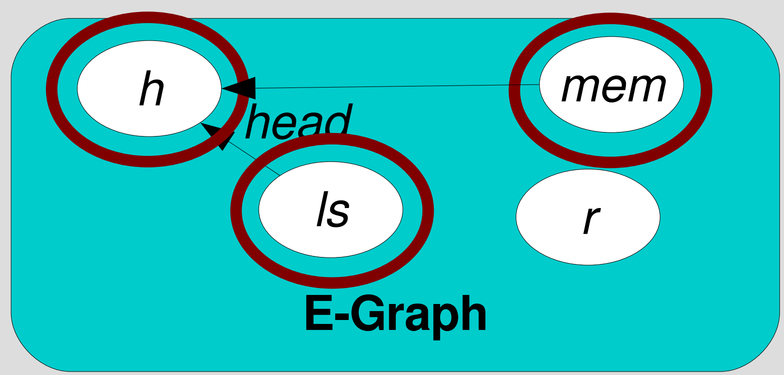
$\boldsymbol{H6: reach(mem, ls, ls)}$

$\boldsymbol{H7: reach(mem, ls, ls)}$

$\boldsymbol{H7: h \geq 0}$

$\boldsymbol{H7: \neg reach(mem, ls, ls) \vee ls = null \vee h \geq 0}$

Prove *False*



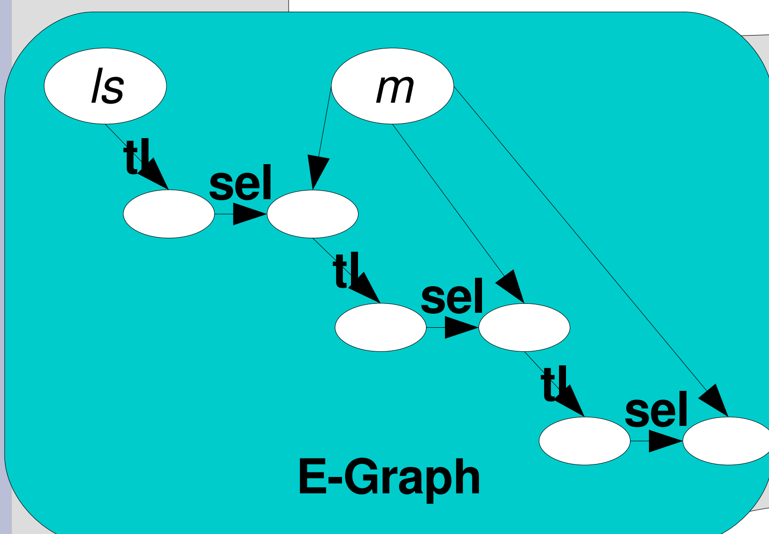
$(\forall x, reach(mem, ls, x) \wedge x \neq null \rightarrow hd(mem, x) \geq 0) \wedge ls \neq null \wedge r \geq 0 \rightarrow hd(mem, ls) + r \geq 0$

Let's Try Another....

$reach(mem, ls, null) \wedge reach(mem, ls, mid) \wedge reach(mem, new, null)$

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

$reach(mem', ls, null)$



Prove *False*

Axiom:

$\forall m, \forall x, \forall y, reach(m,x,y) \rightarrow x = y \vee reach(m, sel(m, tl(x)), y)$

~~**When** any *n*, *u*, and *v* are in the E-Graph~~
~~**Instantiate with** *m* := *n*, *x* := *u*, and *y* := *v*~~

When any *sel*(*n*, *tl*(*v*)) is in the E-Graph
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....

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

$H1 : reach(mem, ls, null)$

$H2 : reach(mem, ls, mid)$

$H3 : reach(mem, new, null)$

$reach(upd(mem, tl(mid), new), ls, null)$

Proof complete!

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

Inductive reac…
| reach_e…
…h_r…

…each mem…

Calls a Nelson-Oppen prover

induction H2.
kettle.
destruct H1; kettle.

Kettle was unable to prove the goal.

The Initial Attempt

$\boldsymbol{H4: tl(ls) = new}$

$\boldsymbol{H5: sel(upd(mem, tl(mid), new), tl(ls)) = new}$

$IH: reach(upd(mem \ldots$ / $H3: reach(mem, del(me \ldots , null)$

$H1: reach(mem, ls \ldots$

$H2 \ldots$

Kettle decided to do a case split on the equality of *tl*(*ls*) ... ere 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 *preserve_reach*.....

```
induction H2.
kettle.
destruct H1; kettle.
use (preserve_reach mem new (tl mid) new); kettle.
```

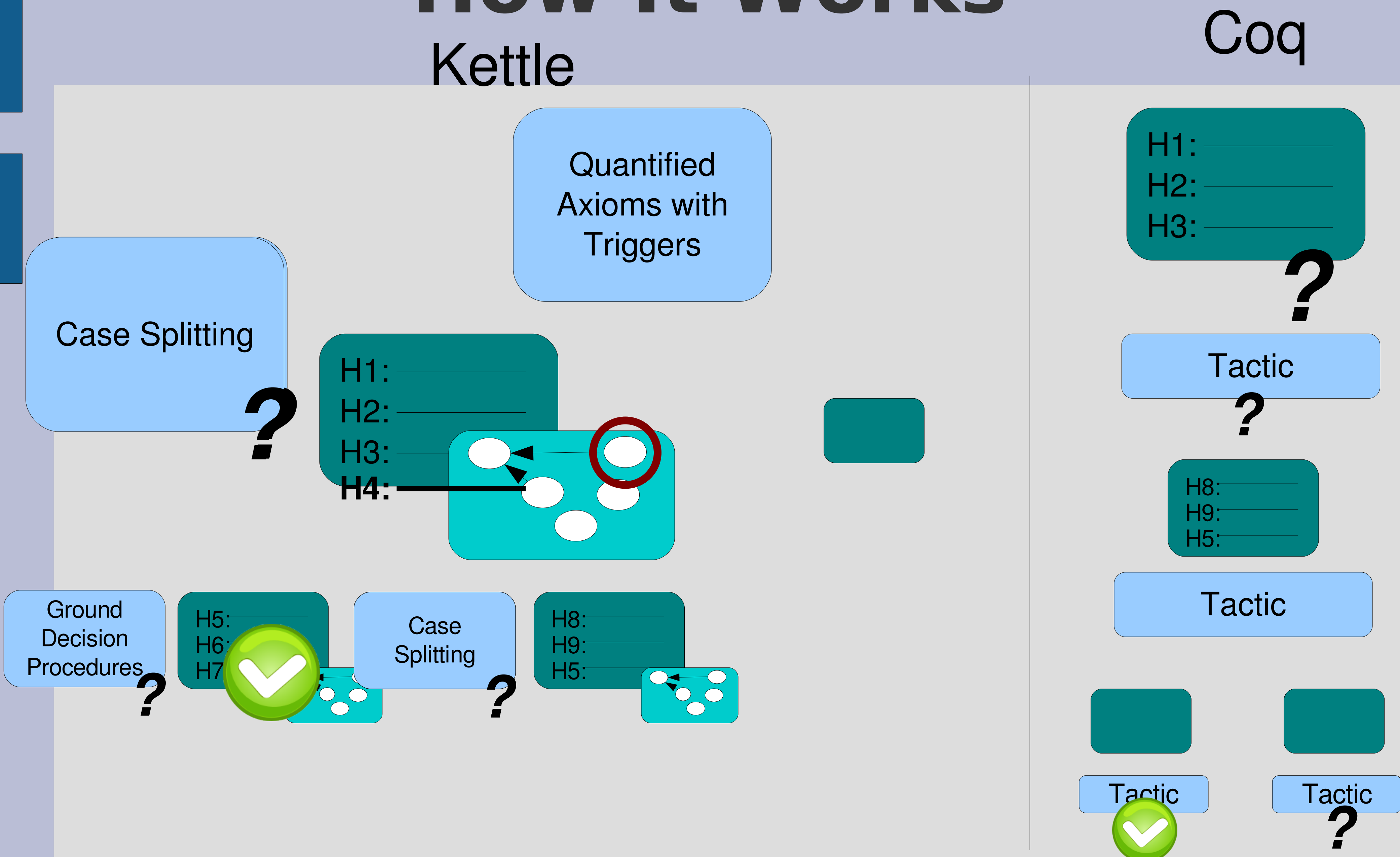
An Even Better Way

- 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'.
```

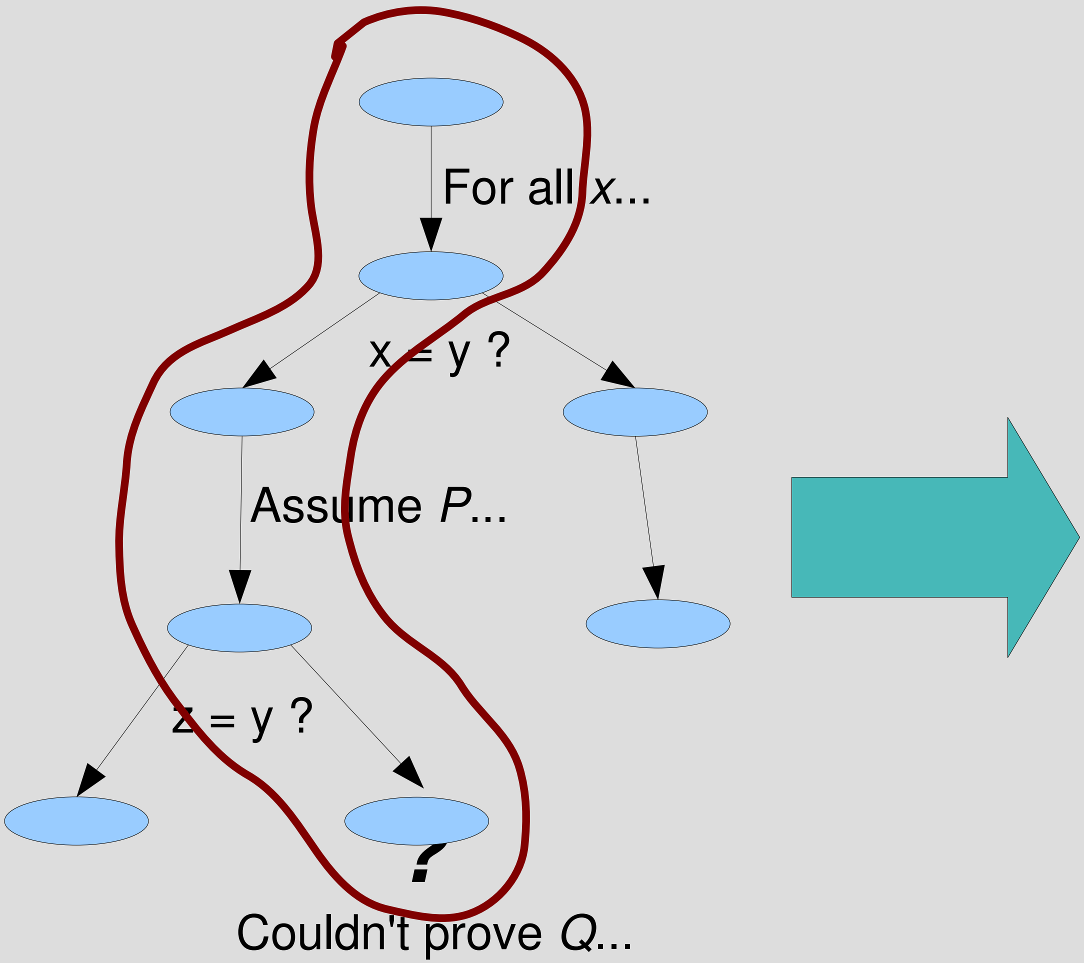
```
induction H2; bounded_kettle.
```

How It Works



Kettle
Coq
Quantified Axioms with Triggers
Case Splitting
H1:
H2:
H3:
H4:
Ground Decision Procedures
H5:
H6:
H7
Case Splitting
H8:
H9:
H5:
H1:
H2:
H3:
Tactic
H8:
H9:
H5:
Tactic
Tactic
Tactic

Proof Translation



$$\forall x, x = y \rightarrow P \rightarrow z = y \rightarrow Q$$

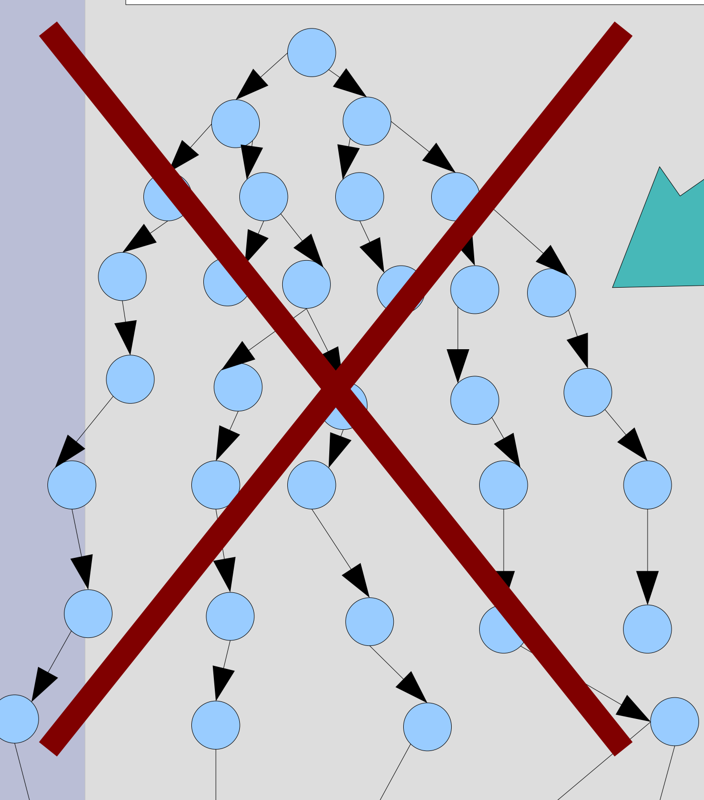
Reflective Proof Checking

"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..."

?

λ

Term in
Dependent Type Theory



```
Definition kettle_prop : Set := ....
Definition kettle_proof : kettle_prop -> Set := ....
Definition interp_prop : kettle_prop -> Prop := ....
Definition check :
    forall (p : kettle_prop) (pf : kettle_proof p),
        interp_prop p := ....
```

Proof

The goal is true because I have a
Kettle proof of a proposition that
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