Interactive Computer Theorem Proving

Lecture 11: Proof by Reflection

CS294-9
November 2, 2006
Adam Chlipala
UC Berkeley
Proofs Revisited

$1 + 1 = 2$

...in standard first-order logic

---

$orall n, m, S \ n + m = n + S \ m$

$\forall E$

$orall m, S \ O + m = O + S \ m$

$\forall E$

---

$S \ O + S \ O = S \ O + S \ O$

$I$

---

$S \ O + S \ O = O + S \ (S \ O)$

$E$

---

$S \ O + S \ O = O + S \ (S \ O)$

$E$

---

$O + S \ (S \ O) = S \ (S \ O)$

$E$

---

$S \ O + S \ O = S \ (S \ O)$
Proofs Revisited

\[ 1 + 1 = 2 \]

...in Coq

Evaluate the proof fully before checking correct rule usage....

\[ S \ O + S \ O = S (S \ O) \]
\[ O + S (S \ O) = S (S \ O) \]
\[ S (S \ O) = S (S \ O) \]

OK, now it's clear that reflexivity justifies this!

\[ S \ O + S \ O = S (S \ O) \]
A Simple Motivating Example

To prove isEven (2n):

\[
\text{Even}_{\text{SS}} (\text{Even}_{\text{SS}} (\text{Even}_{\text{SS}} (\text{Even}_{\text{SS}} (\text{Even}_{\text{SS}} (...\text{Even}_{\text{O}}...))))))
\]

\(n\) Even_{SS}'s

In the real representation, the variable \(n\) of Even_{SS} appears explicitly in each rule invocation, so we have a quadratic-size proof scheme!

**Challenge**: What proof scheme allows us to build isEven proofs for constants such that there exists some constant \(K\) where the proof size for \(2n\) is only \(K\) more than the size of the representation of \(2n\)?
Reflection Recipe (First Cut)

To prove goals that use a predicate $P$:

1. Implement in Coq a (partial) decision procedure for $P$ that approximates the truth of $P$ with an algorithm returning booleans.

2. Prove that $P$ holds when the procedure returns true.

3. Prove individual instances by applying that soundness theorem with reflexivity proofs.
An isEven Decision Procedure

**Fixpoint** check_even \( (n : \text{nat}) : \text{bool} := \)

**match** \( n \) **with**

| 0 => true |
| 1 => false |
| S (S \( n' \)) => check_even \( n' \) |

**end.**

**Key property:** For any constant \( n \), check_even \( n \) evaluates to true or false, using only Coq's built-in reduction rules.
Soundness Theorem

**Theorem** \( \text{check\_even\_sound} : \forall n, \)  
\( \text{check\_even} \ n = \text{true} \rightarrow \text{isEven} \ n. \)

**Generic proof of** \( \text{isEven} \ 2n: \)

\( \text{check\_even\_sound} \ 2n \ (\text{refl\_equal} \ \text{true}) \)

\( \text{check\_even\_sound} \ 2n : \)  
\( \text{check\_even} \ 2n = \text{true} \rightarrow \text{isEven} \ 2n \)

Reduce to:  
\( \text{true} = \text{true} \rightarrow \text{isEven} \ 2n \)
Reflective “Tauto”

\[(P_1 \lor Q_1) \land (P_2 \lor Q_2) \land ... \land (P_n \lor Q_n) \Rightarrow (Q_1 \lor P_1) \land (Q_2 \lor P_2) \land ... \land (Q_n \lor P_n)\]

The tauto tactic (which is also used by intuition) solves this goal by expanding out all \(2^n\) cases arising from the disjunctions in the assumption, leading to exponentially-sized proofs.

**Challenge:** Develop a reflective proof scheme that lets us prove formulas in a useful class of tautologies with proof size **quadratic** in the formula's length.
First Attempt

To keep it simple, let's start by considering formulas built from True, False, and, and or.

\[
\textbf{Fixpoint} \ \text{eval\_formula} \ (P : \text{Prop}) : \text{bool} := \\
\begin{cases}
\text{match } P \text{ with } \\
| \text{True} => \text{true} \\
| \text{False} => \text{false} \\
| P1 \land P2 => \text{eval\_formula } P1 \land \text{eval\_formula } P2 \\
| P1 \lor P2 => \text{eval\_formula } P1 \lor \text{eval\_formula } P2 \\
| _ => \text{false} \\
\end{cases}
\end{align*}

\textit{Prop} isn't an inductive type!
Revised Reflection Recipe

To prove goals that use a predicate $P$:

1. Create a **syntactic** representation $S$ of $P$'s domain $D$.

2. Define a **compilation** of $S$ into $D$.

3. Implement a decision procedure over $S$ and prove that, when it returns **true** for $s$ in $S$, $P$ holds of the **compilation** of $s$.

4. Use the soundness theorem reflectively.
A Syntactic Representation

**Inductive** formula : **Set** :=

| Truth : formula |
| Falsehood : formula |
| And : formula -> formula -> formula |
| Or : formula -> formula -> formula. |

**Fixpoint** interp_formula (f : formula) : **Prop** :=

**match** f **with**

| Truth => True |
| Falsehood => False |
| And f1 f2 => interp_formula f1 \ interp_formula f2 |
| Or f1 f2 => interp_formula f1 \ interp_formula f2 |

**end.**
A Prover

\textbf{Fixpoint} eval\_formula (P : formula) : bool :=
match P with
| Truth => true
| Falsehood => false
| And P1 P2 => eval\_formula P1 && eval\_formula P2
| Or P1 P2 => eval\_formula P1 || eval\_formula P2
end.
The Soundness Theorem

**Theorem** eval_formula_sound : \( \forall (f : \text{formula}), \)
\[ \text{eval}_\text{formula} \ f = \text{true} \]
\[ \rightarrow \text{compile}_\text{formula} \ f. \]

**Generic scheme for proving formula \( P \):**

Compute the **representation** \( f \) of \( P \).
(This step can't be done in Coq's logic!)

Use the proof term:
\[ \text{eval}_\text{formula}_\text{sound} \ f (\text{refl}_\text{equal} \ \text{true}) \]
An Example

Want to prove:

True \lor False

Representation:

Or Truth Falsehood

Proof term:

\[
\begin{align*}
\text{eval_formula_sound} \left( \text{Or Truth Falsehood} \right) & \left( \text{refl_equal true} \right) \\
\text{eval_formula} \left( \text{Or Truth Falsehood} \right) & = true \\
\Rightarrow & \text{compile_formula} \left( \text{Or Truth Falsehood} \right) \\
\text{true} & = true \\
\Rightarrow & \text{compile_formula} \left( \text{Or Truth Falsehood} \right)
\end{align*}
\]
Now let's expand our class of formulas to include, in addition to True, False, and, and or:

• Implication
• Arbitrary propositions as “propositional variables”

\[
\textbf{Type}
\]

\[
\text{Inductive formula} : \text{Set} :=
\]

| Truth : formula |
| Falsehood : formula |
| And : formula -> formula -> formula |
| Or : formula -> formula -> formula |
| Imp : formula -> formula -> formula |
| Atomic : \textbf{Prop} -> formula. |
Not Quite There....

Now we can prove some theorems:

**Definition** easy_prover (f : formula) : bool :=
\[
\text{match } f \text{ with } \\
| \text{Or True } _ => \text{ true } \\
| _ => \text{ false } \\
\text{end.}
\]

But we get stuck for other “easy” ones:

**Example**: How can we write a prover that can show:
\[ P \rightarrow P \]
for any \( P \)?

Problem: We can't compare Props for equality
**Inductive** formula : **Set** :=

- Truth : formula
- Falsehood : formula
- And : formula -> formula -> formula
- Or : formula -> formula -> formula
- Imp : formula -> formula -> formula
- Atomic : var -> formula.

On the side, maintain a mapping:

\[\text{atomics} : \text{var} \rightarrow \text{Prop}\]

We will depend on decidable equality for var:

\[\text{var_eq_dec} : \forall (x\ y : \text{var}), \{x = y\} + \{x <> y\}\]
**It Works!**

Fixpoint interp_formula (f : formula) : Prop :=

    match f with
    | Truth => True
    | And f1 f2 => interp_formula f1 \ interp_formula f2
    | Atomic v => atomics v

    ...

end.

Fixpoint prover (f : formula) : bool :=

    match f with
    | Imp (Atomic v1) (Atomic v2) =>
        if var_eq_dec v1 v2
        then true
        else false

    ...

end.