CSCI 740 - Programming Language Theory

Lecture 9

Introduction to Coq

Instructor: Hossein Hojjat

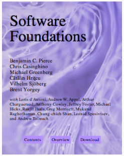
September 20, 2017

Resources

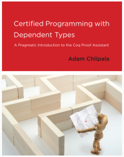
- There is an entire course about Coq at RIT

  "Mechanized Meta-Theory for Programming Language (CSCI-799)"

Useful References



- "Software Foundations" by Pierce et al.
- `http://www.cis.upenn.edu/~bcpierce/sf/`



- "Certified Programming with Dependent Types" by Adam Chlipala
- `http://adam.chlipala.net/cpdt/`

Useful Proof Methods

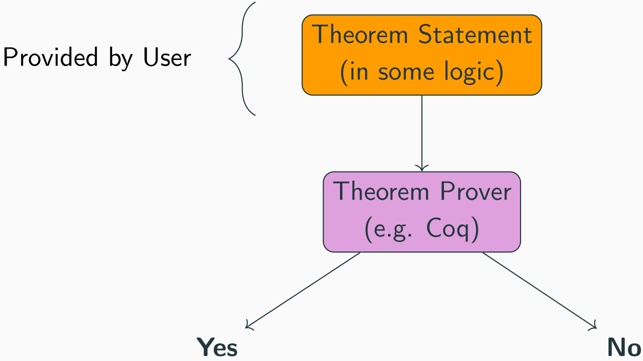
- Proof by example
  - The author gives only the case $n = 2$ and suggests that it contains most of the ideas of the general proof
- Proof by intimidation
  - "Trivial" or "obvious."
- Proof by vigorous handwaving
  - Works well in a classroom, seminar, or workshop setting.
- Proof by cumbersome notation
  - Best done with access to at least four alphabets, special symbols, and the newest release of LaTeX.
- Proof by forward reference
  - Reference is usually to a forthcoming paper of the author, which is often not as forthcoming as at first.

We do not use any of these proof methods in this course!

Selected from `http://mfleck.cs.illinois.edu/proof.html`

Theorem Provers

- Computer programs that can generate and check mathematical theorems
- Theorems are expressed in some mathematical logic
  - such as propositional logic, predicate logic, first-order logic, ...



Proof Checking vs. Proof Generation

- A formal proof is a list of formulas each of which is justified by an axiom or an inference rule applied to earlier formulas

Formulas	Justification
f_1	Axiom
f_2	Rule 2 and f_1
f_3	Rule 4 and f_2
$\cdots$	$\cdots$
Theorem	$\cdots$

- Formal proofs are easy to check mechanically:
  Just make sure the justifications are applied correctly
- Proof generation is hard:
  - Generate a list of formulas, each of which has valid justification
  - Last formula should be the desired theorem

User Input

- What does the user provide?
- Statement of theorem expressed in the logic of the system
- For fully automatic provers, this is enough:
  all you do is give your theorem and push "Go"
  - Example: SPASS, ACL2, Princess
- Semi-automatic provers require user intervention to guide the proof
  - Example: Coq, Isabelle
- Fully automatic provers can't prove as many theorems as "semi-automatic" provers

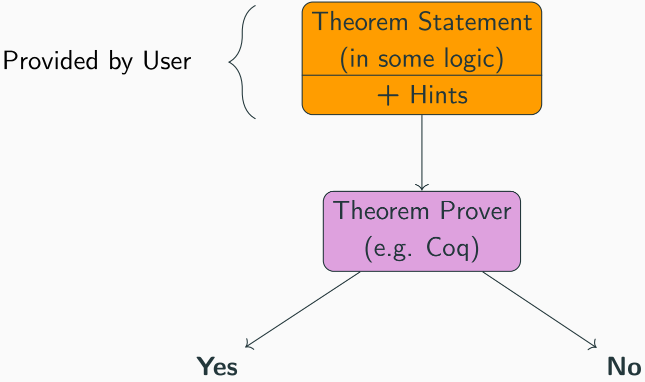
User Input

- User must provide some kind of hints to help the semi-automatic prover
    - Often provided in the same file
- Least useful hint: "A proof exists - search forever until you find it"
- Most useful hint: "Here is the proof: ..."

Most semi-automatic provers take a middle path and require hints between the two extremes

- Statement of key lemmas (useful intermediate results)
- Proof outline (how the lemmas connect)
- Key idea in proof (prove by induction on n)
- Proof script (list of medium-sized steps in the proof)

Theorem Prover Overview



Library

Many theorems share commonly used definitions and lemmas

Examples.

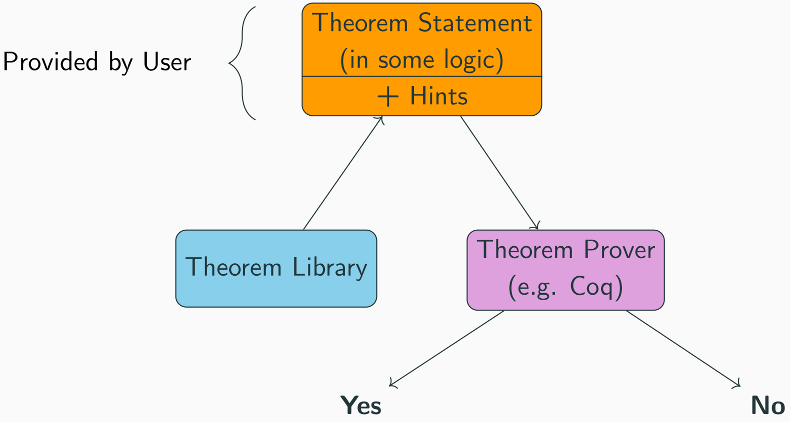
Naturals

- Definition (zero & successor)
- Facts about naturals (e.g., $a + b = b + a$) and their proofs

Integers

- Definition (naturals & negative naturals & zero)
- Facts about integers (e.g., $a + (a) = 0$) and their proofs

Theorem Prover Overview



Coq

- Coq is a proof assistant developed since 1984 by INRIA (France)
- Calculus of Inductive Constructions is the formalism behind Coq
  - Typed lambda-calculus with dependent types and inductive types
- French have a tradition of naming their software as animal species (e.g. Caml)
- In French, “coq” means rooster and it sounds like the initials of the Calculus of Constructions (CoC)

Coq: Popular & Successful

- Research project lead by a group of universities
  - Penn, MIT, Yale, Princeton
- Push forward the state of the art in applying Coq to verify realistic software and hardware
  - `https://deepspec.org/`

Coq: Popular & Successful

- CompCert: a realistic, industrially-usable compiler for the C language
- http://compcert.inria.fr/
- 100k lines of Coq and 6 person-years of effort
- Among the largest ever proof performed with a proof assistant

"Closing the gap - The formally verified compiler CompCert", (SSS 2017)

Finding and Understanding Bugs in C Compilers

Xuejun Yang   Yang Chen   Eric Eide   John Regehr
University of Utah, School of Computing
{jxyang, chenyang, eeide, regehr}@cs.utah.edu

[PLDI'11]

"The striking thing about our CompCert results is that the middle end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task. The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users."

Tactic-based System

- Calculus of Inductive Constructions is quite powerful: automatic proof generation is quite limited
- Instead, user provides hints in the form of proof scripts
- Proof scripts:
  lists of tactics, which guide Coq in generating the proof

Key Steps

- Introduce definitions and theorems
- Prove them by applying deductive steps called tactics
- Coq Tactics Quick Reference
  `http://adam.chlipala.net/itp/tactic-reference.html`
- Tactics Index
  `https://coq.inria.fr/refman/tactic-index.html`

Proof: On Paper

Theorem: For all n, $\quad 2 \sum_{0 \leq i \leq n} i = n(n+1)$

Proof: By induction on n.

<u>Base case:</u> $(n = 0)$:

$$2 \sum_{0 \leq i \leq n} i = 0 = 0(0+1)$$

<u>Inductive case:</u> $(n = n' + 1)$:

IH: $\quad 2 \sum_{0 \leq i \leq n'} i = n'(n'+1)$

$$\begin{aligned} 2 \sum_{0 \leq i \leq n} i &= 2(n'+1) + 2 \sum_{0 \leq i \leq n'} i \\ &= 2(n'+1) + n'(n'+1) \\ &= (n'+1)(n'+2) \end{aligned}$$

Proof: In Coq

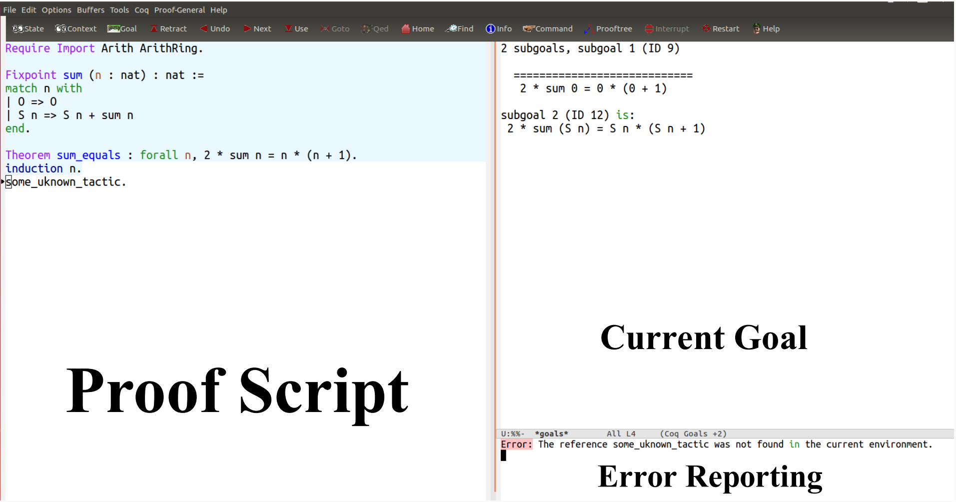
```
Require Import Arith ArithRing.

Fixpoint sum (n : nat) : nat :=
match n with
| O => O
| S n => S n + sum n
end.
```

Just a familiar ADT and Recursive Function Definition

```
Theorem sum_equals : forall n, 2 * sum n = n * (n + 1).
induction n.
trivial.
cbv [sum].
rewrite mult_plus_distr_l.
fold sum.
rewrite IHn.
ring.
Qed.
```

Sequence of tactics

Proof: In Coq


File Edit Options Buffers Tools Coq Proof-General Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Require Import Arith ArithRing.
Fixpoint sum (n : nat) : nat :=
match n with
| O => O
| S n => S n + sum n
end.
Theorem sum_equals : forall n, 2 * sum n = n * (n + 1).
induction n.
some_uknown_tactic.
2 subgoals, subgoal 1 (ID 9)
============================
2 * sum 0 = 0 * (0 + 1)
subgoal 2 (ID 12) is:
2 * sum (S n) = S n * (S n + 1)
U:%%- *goals* All L4 (Coq Goals +2)
Error: The reference some_uknown_tactic was not found in the current environment.
Proof Script
Current Goal
Error Reporting

Proving Theorems with Tactics

- Basic syntax to introduce lemmas and theorems

```
Lemma O_plus : forall n, plus O n = n.
Proof.
  (* Sequence of tactics *)
Qed.
```

- Lemma and Theorem are interchangeable
  (You can also say Remark, Corollary, Fact or Proposition)

Tactics

- Tactics instruct Coq on the steps to take to prove a theorem

Reflexivity

- Prove an equality goal that follows by normalizing terms

Induction x

- Prove goal by induction on quantified variable x
- Structural Induction: x is any recursively defined
- All variables appearing before x will remain fixed throughout the induction

Tactics

simpl

- Apply standard heuristics for computational simplification in conclusion
- Often involves doing some β-reductions

rewrite H

- Use (potentially quantified) equality H to rewrite in the conclusion

intros

- Move quantified variables and/or hypotheses above the double line

apply thm

- Apply a named theorem, reducing the goal into one new subgoal for each of the theorem's hypotheses, if any

Tactics

assumption

- Prove a conclusion that matches a known hypothesis

destruct E

- Do case analysis on the constructor used to build term E

Next Session

Bring a laptop that has Coq and Proof General already installed
(if you have access to a laptop)