Automating Interactive Theorem Proving with Coq and Ltac

by Oron Propp and Alex Sekula
Mentored by Drew Haven
PRIMES

Motivation

- Math is usually written by hand, checked by other mathematicians

- Verifying process takes time out of learning

- Computer potentially gives instant feedback

Overview

- Coq – a language that verifies proofs

- Goal
  - Use Coq to help students learn proving, allowing them to check their proofs by themselves

- Problem
  - Coq is difficult to learn, and "raw Coq code" looks nothing like the original proof

"Raw Coq code" Example

```
specialize(H1 _ H9).
intuition.
intuition.
rewrite H4 in H3.
destruct H2 as [t].
destruct H2 as [j].
assert(Z.divide x a).
unfold Z.divide.
exists q.
intuition.
rewrite H5.
ring.
```

A Brief Intro to Coq

- Fun Fact: Coq means rooster in French!
  - Developed in INRIA, France

- A basic Coq proof consists of:
  - Theorem statements
  - Tactics

- Environment
  - Coq code
  - List of hypotheses
  - List of subgoals

Approach

- Wrote number theory proofs in Coq

Two column proof → "Raw" Coq proof

Look for repetitive, messy, trivial parts

Develop approach to eliminate these parts

Automate Coq Proof

Approach

Proofs written:

- Well ordering principle for natural numbers
- No natural number between 0 and 1
- Infinitude of primes
  - Fundamental theorem of arithmetic
- Division algorithm
- Extended Euclidean algorithm
- Euclidean algorithm

Two Column "Math" Proof

	Theorem: there does not exist a natural number n such that $0 < n < 1$	
1	Let S be the set of all natural numbers between 0 and 1	Definition
2	Assume that S is non-empty	For sake of contradiction
3	Let r be the least element of S	Well ordering principle and 2
4	$0 < r < 1$	r is in S
5	$0 * r < r * r < 1 * r$	4 and math
6	$0 < r^2 < r$	5 and math
7	$r^2 < 1$	4 and 6
8	r^2 is in S	6, 7 and 1
9	$r^2 < r$	6
10	Contradiction	3 and 9
11	Since S has no least element, it is empty, and our theorem is true	Definition
12	QED	

Ugly Duckling

```
Theorem no_nat_between_1_and_0 : ~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
  destruct (nat_well_ordered _ H).
  clear H.
  destruct H0.
  assert (0 * x < x * x < 1 * x).
  split; apply mult_lt_compat_r; apply H.
  assert (0 < x * x < x).
  replace 0 with (0 * x) by auto with arith.
  replace (1 * x) with x in H1 by auto with arith.
  assumption.
  assert (x * x < 1).
  apply lt_trans with x.
  apply H2.
  apply H.
  assert (0 < x * x < 1).
  split.
  apply H2.
  assumption.
  specialize (H0 _ H4).
  apply (le_not_lt _ _ H0).
  apply H2.
Qed.
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.
```

```
============================
 ~ (exists n : nat, 0 < n < 1)
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
```

```
  H : exists n : nat, 0 < n < 1
  ============================
   False
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
  destruct (nat_well_ordered _
H).
```

```
  H : exists n : nat, 0 < n < 1
  x : nat
  H0 : 0 < x < 1 /\ (forall a :
nat, 0 < a < 1 -> x <= a)
  ============================
   False
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
  destruct (nat_well_ordered _
H).
  clear H.
```

```
  x : nat
  H0 : 0 < x < 1 /\ (forall a :
nat, 0 < a < 1 -> x <= a)
  ============================
   False
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
  destruct (nat_well_ordered _
H).
  clear H.
  destruct H0.
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  ============================
   False
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
  destruct (nat_well_ordered _
H).
  clear H.
  destruct H0.
  assert (0 * x < x * x < 1 *
x).
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  ============================
   0 * x < x * x < 1 * x

subgoal 2 (ID 1226) is:
 False
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
  destruct (nat_well_ordered _
H).
  clear H.
  destruct H0.
  assert (0 * x < x * x < 1 *
x).
  split; apply mult_lt_compat_r;
apply H.
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  H1 : 0 * x < x * x < 1 * x
  ============================
   False
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
  destruct (nat_well_ordered _
H).
  clear H.
  destruct H0.
  assert (0 * x < x * x < 1 *
x).
  split; apply mult_lt_compat_r;
apply H.
  assert (0 < x * x < x).
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  H1 : 0 * x < x * x < 1 * x
  ============================
   0 < x * x < x

subgoal 2 (ID 1249) is:
 False
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
  destruct (nat_well_ordered _
H).
  clear H.
  destruct H0.
  assert (0 * x < x * x < 1 *
x).
  split; apply mult_lt_compat_r;
apply H.
  assert (0 < x * x < x).
  replace 0 with (0 * x) by auto
with arith.
  replace (1 * x) with x in H1
by auto with arith.
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  H1 : 0 * x < x * x < x
  ============================
   0 * x < x * x < x

subgoal 2 (ID 1249) is:
 False
```

Proving With Coq

```
Theorem no_nat_between_1_and_0 :
~ exists n : nat, 0 < n < 1.

Proof.
  intro H.
  destruct (nat_well_ordered _
H).
  clear H.
  destruct H0.
  assert (0 * x < x * x < 1 *
x).
  split; apply mult_lt_compat_r;
apply H.
  assert (0 < x * x < x).
  replace 0 with (0 * x) by auto
with arith.
  replace (1 * x) with x in H1
by auto with arith.
  assumption.
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  H1 : 0 * x < x * x < 1 * x
  H2 : 0 < x * x < x
  ============================
   False
```

Proving With Coq

```
  assumption.
  assert (x * x < 1).
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  H1 : 0 * x < x * x < 1 * x
  H2 : 0 < x * x < x
  ============================
   x * x < 1

subgoal 2 (ID 1264) is:
 False
```

Proving With Coq

```
  assumption.
  assert (x * x < 1).
  apply lt_trans with x.
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  H1 : 0 * x < x * x < 1 * x
  H2 : 0 < x * x < x
  ============================
   x * x < x

subgoal 2 (ID 1266) is:
 x < 1
subgoal 3 (ID 1264) is:
 False
```

Proving With Coq

```
assumption.
assert (x * x < 1).
apply lt_trans with x.
apply H2.
apply H.
```

```
x : nat
H : 0 < x < 1
H0 : forall a : nat, 0 < a < 1
-> x <= a
H1 : 0 * x < x * x < 1 * x
H2 : 0 < x * x < x
H3 : x * x < 1
============================
 False
```

Proving With Coq

```
  assumption.
  assert (x * x < 1).
  apply lt_trans with x.
  apply H2.
  apply H.
  assert (0 < x * x < 1).
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  H1 : 0 * x < x * x < 1 * x
  H2 : 0 < x * x < x
  H3 : x * x < 1
  ============================
   0 < x * x < 1

subgoal 2 (ID 1272) is:
 False
```

Proving With Coq

```
assumption.
assert (x * x < 1).
apply lt_trans with x.
apply H2.
apply H.
assert (0 < x * x < 1).
split.
```

```
  x : nat
  H : 0 < x < 1
  H0 : forall a : nat, 0 < a < 1
-> x <= a
  H1 : 0 * x < x * x < 1 * x
  H2 : 0 < x * x < x
  H3 : x * x < 1
  ============================
   0 < x * x

subgoal 2 (ID 1275) is:
 x * x < 1
subgoal 3 (ID 1272) is:
 False
```

Proving With Coq

```
assumption.
assert (x * x < 1).
apply lt_trans with x.
apply H2.
apply H.
assert (0 < x * x < 1).
split.
apply H2.
assumption.
```

```
x : nat
H : 0 < x < 1
H0 : forall a : nat, 0 < a < 1
-> x <= a
H1 : 0 * x < x * x < 1 * x
H2 : 0 < x * x < x
H3 : x * x < 1
H4 : 0 < x * x < 1
============================
 False
```

Proving With Coq

```
assumption.
assert (x * x < 1).
apply lt_trans with x.
apply H2.
apply H.
assert (0 < x * x < 1).
split.
apply H2.
assumption.
specialize (H0 _ H4).
```

```
x : nat
H : 0 < x < 1
H0 : x <= x * x
H1 : 0 * x < x * x < 1 * x
H2 : 0 < x * x < x
H3 : x * x < 1
H4 : 0 < x * x < 1
============================
 False
```

Proving With Coq

```
assumption.
assert (x * x < 1).
apply lt_trans with x.
apply H2.
apply H.
assert (0 < x * x < 1).
split.
apply H2.
assumption.
specialize (H0 _ H4).
apply (le_not_lt _ _ H0).
```

```
x : nat
H : 0 < x < 1
H0 : x <= x * x
H1 : 0 * x < x * x < 1 * x
H2 : 0 < x * x < x
H3 : x * x < 1
H4 : 0 < x * x < 1
============================
 x * x < x
```

Proving With Coq

```
assumption.
assert (x * x < 1).
apply lt_trans with x.
apply H2.
apply H.
assert (0 < x * x < 1).
split.
apply H2.
assumption.
specialize (H0 _ H4).
apply (le_not_lt _ _ H0).
apply H2.
```

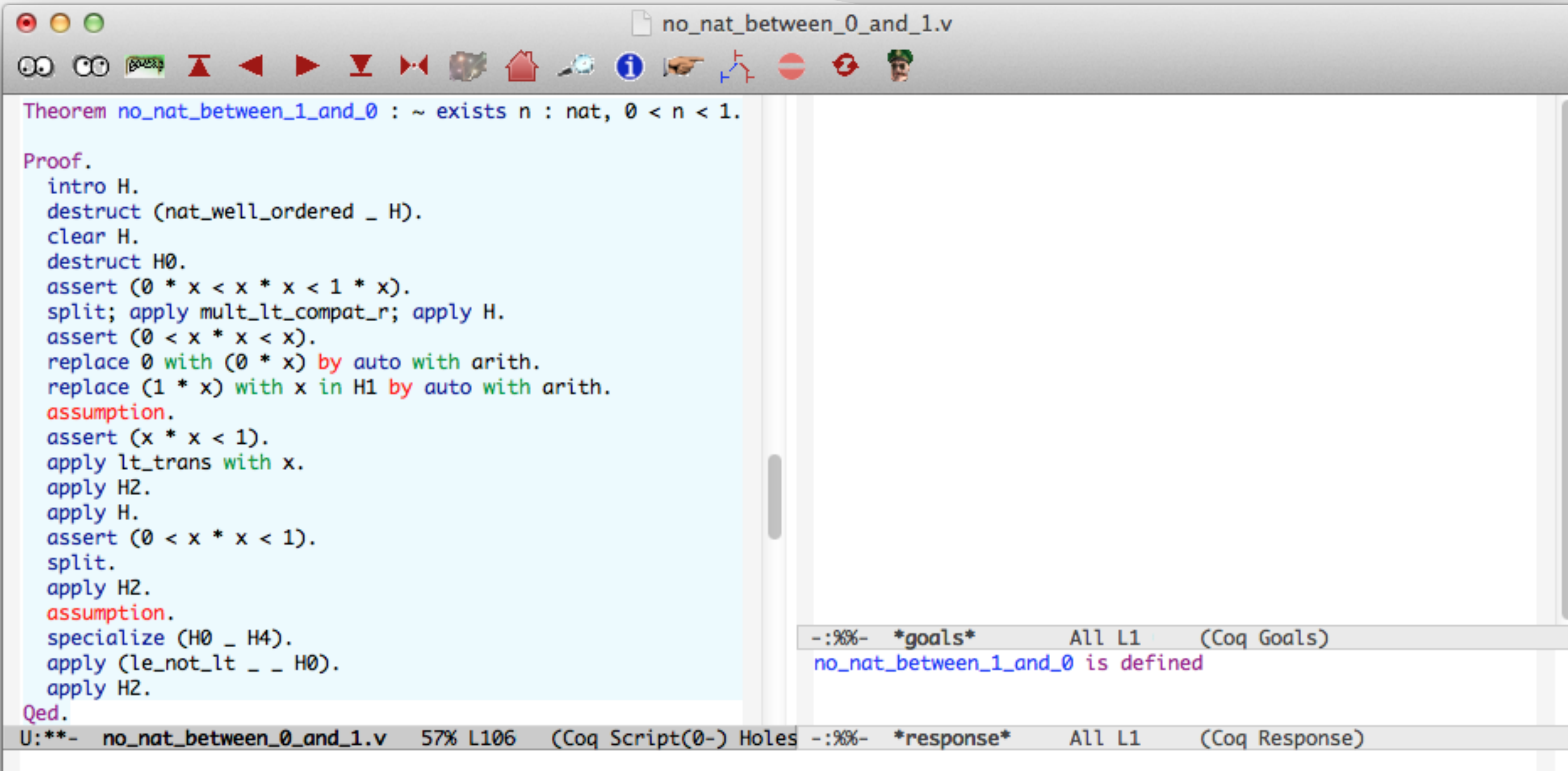
```
No more subgoals.
(dependent evars:)
```

Proving With Coq

```
  assumption.
  assert (x * x < 1).
  apply lt_trans with x.
  apply H2.
  apply H.
  assert (0 < x * x < 1).
  split.
  apply H2.
  assumption.
  specialize (H0 _ H4).
  apply (le_not_lt _ _ H0).
  apply H2.
Qed.
```

```
no_nat_between_1_and_0 is
defined
```

Proving With Coq



Proof Automation

- Noticed repeated, tedious code

- Goal - less cluttered code, easier to write
- Write tactics that automate trivial steps

- Three tactics written
  - Zwop
  - math
  - simplify

"Zwop" Tactic

- Applies the well-ordering principle (WOP) to a set in Coq
- WOP is a key part of many number theory proofs

```
Ltac Zwop A Adef :=
  pose(A := Adef);
  destruct (bounded_Z_well_ordered A).
```

"math" Tactic

- Tries to simplify ring inequalities and uses "intuition" tactic

```
Ltac math :=
  intuition;
    repeat match goal with
            | [ H : ?T, H' : ?T' |- _ ] =>
              match type of T with
                | Prop => match type of T' with
                            | Prop => assert (T /\ T') by tauto; clear H H'
                          end
              end
          end;
    repeat match goal with
            | [ H : context[?a < ?b] |- _ ] => progress ring_simplify a b in H;
 progress apply lt_trans with a in H; progress apply lt_trans with b in H
          end;
    intuition.
```

"simplify" Tactic

- Tries many general, useful tactics where applicable
- Only uses them if they advance the proof

```
Ltac simplify_step v :=
  repeat (try match goal with
          | [ |- _ /\ _] => split
          | [ H : _ /\ _ |- _ ] => destruct H
          | [ |- _ = _] => ring
          | [ H : exists _, _ |- _] => let v' := fresh v in destruct H as [v']

          | [|- exists _, _] => eauto with arith
          end;
  intuition;
  simpl in *;
  unfold Z.divide in *).

Ltac simplify' v :=
  simplify_step v;
  try now (subst; simplify_step v).

Tactic Notation "simplify" := let v := fresh "v" in simplify' v.

Tactic Notation "simplify" ident(v) := simplify' v.
```

The Power of "simplify"

From this...

```
Lemma remainder_not_0 : forall a b, remainder a b = 0 -> a <> 0 -> b <> 0.

Proof.
  intros.
  apply NNPP; intro.
  assert (b = 0); intuition.
  subst.
  unfold remainder in H; intuition.
Qed.
```

The Power of "simplify"

To this…

```
Proof.
  simplify.
Qed.
```

Lists From Sets

- Synthesizes list of elements from finite set
- Allows us to compute operations on the set

```
Lemma finite_set_list : forall T (E : Ensemble T), Finite T E ->
exists L : list T, forall t : T, List.In t L <-> Ensembles.In _ E t.
```

```
Fixpoint product_of_list (l : list Z) : Z :=
  match l with
    | nil => 1
    | cons a l' => Zmult a (product_of_list l')
  end.
```

Assertion Technique

- Coq uses backwards proving
- Forward proving is more natural
- Used `assert` tactic to do so

- Three steps
  - Write out two column proof
  - Convert the left column into assertions in the proof
  - Prove each assertion using the ones before it

Basic Frameworks

```
Theorem theorem_name : [theorem_statement].
Proof.
   intros.
   assert([step 1]).
      [code used to prove step 1]
   assert([step 2]).
      [code used to prove step 2]
   ...
Qed.
```

Beautiful Swan

```
Theorem no_nat_between_1_and_0 : ~ exists n : nat, 0 < n < 1.
Proof.
  intro.
  pose (U := (fun n => 0 < n < 1)).
  destruct (nat_well_ordered U) as [r]; [ auto | ].
  unfold U in H0.
  assert (0*r < r*r < 1*r) by (apply mult_double_lt_compat_r; math).
  assert (0 < r*r < r) by math.
  assert (r*r < 1) by (apply lt_trans with r; math).
  assert (r*r ∈ U) by (unfold U; math).
  assert (r*r < r) by math.
  assert (r <= r*r) by math.
  omega.
Qed.
```

Conclusion

- Our findings have educational implications
  - Coq proofs now resemble mathematical proofs more
    - More readable
  - Proofs are easier to write
    - Don't have to do everything out
  - Easier for students to use Coq to learn to write proofs and validate their proofs

Future

- Interpreter
  - Inputs higher level language, outputs Coq code

- Doing out more Number Theory proofs in Coq for more automation

Acknowledgements

- Our mentor Drew Haven
- Our overseeing professor Dr. Adam Chlipala
- PRIMES
- Our parents