Software Verification
(preview of COS 510 “Programming Languages”)

Andrew W. Appel
Princeton University
Formal reasoning about programs

Functional Programming

Proving your (functional) programs correct
Formal reasoning about programs and programming languages

- Intro to Formal Logic
- Functional Programming
- Proving your (functional) programs correct
- Type systems
- Specification of programming languages
- Proving your type system sound
- Imperative Programming
- Hoare Logic
- Proving Hoare Logic sound
- Proving your (imperative) programs correct
Which of these things do we do
By machine?

- Intro to Formal Logic
- Functional Programming

With pencil+paper?

- Specification of programming languages
- Imperative Programming
- Type systems
- Proving Hoare Logic sound
- Proving your type system sound
- Proving your (functional) programs correct
- Proving your (imperative) programs correct
We can do all of these

By machine!

pencil+paper? Really?

Proving your (functional) programs correct

Proving your type system sound

Proving your (imperative) programs correct

Introduction to Formal Logic

Functional Programming

Specification of programming languages

Imperative Programming

Type systems

Hoare Logic

Proving Hoare Logic sound

Proving your type system sound
COS 510: Machine-checked, formal reasoning about programs and programming languages

- Intro to Formal Logic
- Functional Programming
- Proving your (functional) programs correct
- Specification of programming languages
- Type systems
- Proving your type system sound
- Imperative Programming
- Hoare Logic
- Proving Hoare Logic sound
- Proving your (imperative) programs correct
EXAMPLE: LENGTH, APP
Require Import List.

Fixpoint length {A} (xs: list A) : nat :=
  match xs with
  | nil => 0
  | x::xs' => 1 + length xs'
end.

Eval compute in length (1::2::3::4::nil).

Fixpoint app {A} (xs ys: list A) : list A :=
  match xs with
  | nil => ys
  | x::xs' => x :: app xs' ys
end.

Eval compute in app (1::2::3::nil) (7::8::nil).

Eval compute in length (app (1::2::3::nil) (7::8::nil)).
Require Import List.

Fixpoint length {A} (xs: list A) : nat :=
match xs with
| nil => 0
| x::xs' => 1 + length xs'
end.

Eval compute in length (1::2::3::4::nil).

Fixpoint app {A} (xs ys: list A) : list A :=
match xs with
| nil => ys
| x::xs' => x :: app xs' ys
end.

Eval compute in app (1::2::3::nil) (7::8::nil).

Eval compute in length (app (1::2::3::nil) (7::8::nil)).
Require Import List.

Fixpoint length {A} (xs: list A) : nat :=
  match xs with
  | nil => 0
  | x::xs' => 1 + length xs'
end.

Eval compute in length (1::2::3::4::nil).

Fixpoint app {A} (xs ys: list A) : list A :=
  match xs with
  | nil => ys
  | x::xs' => x :: app xs' ys
end.

Eval compute in app (1::2::3::nil) (7::8::nil).

Eval compute in length (app (1::2::3::nil) (7::8::nil)).
Require Import List.

Fixpoint length {A} (xs: list A) : nat :=
  match xs with
  | nil => 0
  | x::xs' => 1 + length xs'
end.

Eval compute in length (1::2::3::4::nil).

Fixpoint app {A} (xs ys: list A) : list A :=
  match xs with
  | nil => ys
  | x::xs' => x :: app xs' ys
end.

Eval compute in app (1::2::3::nil) (7::8::nil).

Eval compute in length (app (1::2::3::nil) (7::8::nil)).
Require Import List.

Fixpoint length {A} (xs : list A) : nat :=
  match xs with
  | nil => 0
  | x :: xs' => 1 + length xs'
end.

Eval compute in length (1 :: 2 :: 3 :: 4 :: nil).

Fixpoint app {A} (xs ys : list A) : list A :=
  match xs with
  | nil => ys
  | x :: xs' => x :: app xs' ys
end.

Eval compute in app (1 :: 2 :: 3 :: nil) (7 :: 8 :: nil).

Eval compute in length (app (1 :: 2 :: 3 :: nil) (7 :: 8 :: nil)).
Theorem app_length: forall {A} (xs ys: list A),
    length (app xs ys) = length xs + length ys.
Proof.
Qed.
Theorem app_length: forall {A} (xs ys: list A), length (app xs ys) = length xs + length ys.

Proof.
Qed.

1 subgoal

forall (A : Type) (xs ys : list A), length (app xs ys) = length xs + length ys
**Theorem** app_length: forall {A} (xs ys : list A), length (app xs ys) = length xs + length ys.

**Proof.**

intros.

Qed.
**Theorem** app_length: forall {A} (xs ys: list A), length (app xs ys) = length xs + length ys.

**Proof.**

intros.
induction xs.

- (* base case *)
  simpl.
  reflexivity.
- (* inductive case *)
  simpl.
  reflexivity.

Qed.
Theorem app_length: forall {A} (xs ys: list A), length (app xs ys) = length xs + length ys.

Proof.
  intros.
  induction xs.
  (* base case *)
  simpl.
  reflexivity.
  (* inductive case *)
  simpl.
  reflexivity.
Qed.
Theorem app_length: forall {A} (xs ys: list A),
  length (app xs ys) = length xs + length ys.
Proof.
intros.
induction xs.
- (* base case *)
simpl.
  reflexivity.
- (* inductive case *)
simpl.
  reflexivity.
Qed.
Theorem app_length: \(\forall A (x : A, y : A)\),
\[\text{length} (\text{app} \ x \ y) = \text{length} x + \text{length} y\]

Proof.
intros.
induction xs.
- (* base case *)
simpl.
reflexivity.
- (* inductive case *)
simpl.
reflexivity.
Qed.
Theorem app_length: \(\forall A \ (xs \ ys : \text{list } A), \) 
\(\text{length } (\text{app } xs \ ys) = \text{length } xs + \text{length } ys.\)

Proof.
intros.
induction xs.
- (* base case *)
simpl.
  reflexivity.
- (* inductive case *)
simpl.
  reflexivity.
Qed.
**Theorem** app_length: forall {A} (xs ys: list A), length (app xs ys) = length xs + length ys.

**Proof.**

intros.
induction xs.
- (* base case *)
simpl.
reflexivity.
- (* inductive case *)
simpl.
reflexivity.
Qed.

This subproof is complete, but there are some unfocused goals:

```
length (app (a :: xs) ys) = length (a :: xs) + length ys
```

Messages
Theorem app_length: forall {A} (xs ys: list A),
  length (app xs ys) = length xs + length ys.

Proof.
intros.
induction xs.
- (* base case *)
  simpl.
  reflexivity.
  [ (* inductive case *)]
  simpl.
  reflexivity.
Qed.
**Theorem** `app_length` : \( \text{forall } \{A\} (xs \text{ ys} : \text{list } A), \text{length } (\text{app } xs \text{ ys}) = \text{length } xs + \text{length } ys. \)

**Proof**.

intros.

induction `xs`.

- (* base case *)
  simpl.
  reflexivity.

- (* inductive case *)
  simpl.
  reflexivity.

Qed.
Theorem app_length: forall {A} (xs ys: list A),
    length (app xs ys) = length xs + length ys.
Proof.
intros.
induction xs.
- (* base case *)
  simpl.
  reflexivity.
- (* inductive case *)
  simpl.
  reflexivity.
Qed.

1 subgoal
A : Type
a : A
xs, ys : list A
IHxs : length (app xs ys) =
    length xs + length ys

S (length (app xs ys)) =
S [length xs + length ys]

In environment
A : Type
a : A
xs, ys : list A
IHxs : length (app xs ys) =
    length xs + length ys
Unable to unify "S (length xs + length ys)"
with "S (length (app xs ys))". 

Ready, proving app_length
Theorem app_length: forall {A} (xs ys: list A),
length (app xs ys) = length xs + length ys.

Proof.
intros.
induction xs.
- (* base case *)
simpl.
reflexivity.
- (* inductive case *)
simpl.
rewrite IHxs.
reflexivity.
Qed.
**Theorem** app_length: forall {A} (xs ys: list A),
  length (app xs ys) = length xs + length ys.

**Proof.**
intros.
induction xs.
- (* base case *)
simpl.
  reflexivity.
- (* inductive case *)
simpl.
  rewrite IHxs.
  reflexivity.
Qed.

No more subgoals.
Theorem app_length: forall {A} (xs ys: list A), length (app xs ys) = length xs + length ys.

Proof.
intros.
induction xs.
- (* base case *)
simpl.
reflexivity.
- (* inductive case *)
simpl.
rewrite IHxs.
reflexivity.
Qed.
Theorem app_assoc: forall {A} (xs ys zs : list A),
  app xs (app ys zs) = app (app xs ys) zs.

Proof.
  intros.
  induction xs.
  - (* base case *)
    simpl.
    reflexivity.
  - (* inductive case *)
    simpl.
    rewrite IHxs.
    reflexivity.
  Qed.
Theorem app_assoc: forall {A} (xs ys zs: list A),
    app xs (app ys zs) = app (app xs ys) zs.

Proof.
 intros.
 induction xs.
 - (* base case *)
   simpl.
   reflexivity.
 - (* inductive case *)
   simpl.
   rewrite IHxs.
   reflexivity.
Qed.
Applications of Formal Methods
Attacking a web server

URLs
Input in web forms
Crypto keys for SSL
etc.

for (i=0; p[i]; i++)
search[i] = p[i];

this is a really long search term that overflows a buffer
Attacking a web browser

HTML keywords
Images
Image names
URLs
etc.

for(i=0;p[i];i++)
gif[i]=p[i];

Client PC

Web Server
@ badguy.com

www.badguy.com

Earn $$$ Thousands working at home!
Attacking everything in sight

E-mail client
PDF viewer
Web browser
Operating-system kernel
TCP/IP stack

Any application that ever sees input directly from the outside
Solution: implement the outward-facing parts of software without any bugs!

E-mail client
PDF viewer
Web browser
Operating-system kernel
TCP/IP stack

Any application that ever sees input directly from the outside
In recent years, great progress in . . .

- Proved-correct optimizing C compiler (France)
- Proved-correct ML compiler (Sweden, Princeton)
- Proved-correct O.S. kernels (Australia, New Haven)
- Proved-correct crypto (Princeton NJ, Cambridge MA)
- Proved-correct distributed systems (Seattle, Israel)
- Proved-correct web server (Philadelphia)
- Proved-correct malloc/free library (Princeton, Hoboken)
Automated verification in industry

Amazon
Microsoft
Intel
Facebook
Google
Galois, HRL, Rockwell, Bedrock, ...
Recent Princeton JIW / Sr. Thesis

- Katherine Ye ’16 verified crypto security
- Naphat Sanguansin ’16 verified crypto impl’n
- Brian McSwiggen ’18 verified B-trees
- Katja Vassilev ’19 verified dead-var elimination
- John Li ’19 verified uncurrying
- Jake Waksbaum ’20 verified Burrows-Wheeler
- Anvay Grover ’20 verified CPS-conversion
Verified Correctness and Security of mbedTLS HMAC-DRBG

Katherine Q. Ye ’16
Princeton U., Carnegie Mellon U.
Lennart Beringer
Princeton University

Matthew Green
Johns Hopkins University

Naphat Sanguansin’16
Princeton University

Adam Petcher
Oracle

Andrew W. Appel ’81
Princeton University

ABSTRACT
We have formalized the functional specification of HMAC-DRBG (NIST 800-90A), and we have proved its cryptographic security—that its output is pseudorandom—using a hybrid game-based proof. We have also proved that the mbedTLS implementation (C program) correctly implements this functional specification. That proof composes with an existing C compiler correctness proof to guarantee, end-to-end, that the machine language program gives strong pseudorandomness. All proofs (hybrid games, C program verification, compiler, and their composition) are machine-checked in the Coq proof assistant. Our proofs are modular: the hybrid game proof holds on any implementation of HMAC-DRBG that satisfies our functional specification. Therefore, our functional specification can serve as a high-assurance reference.
Prerequisites for COS 510 if you’re an undergrad

1. **COS 326 Functional Programming**
2. Enjoy the proofs in **COS 326**
3. Get the form signed by Colleen Kenny-McGinley, room 210 (one-stop shopping, all three signatures):