CS 494 SF – Software Foundations

William Mansky
How can we write programs that we know are correct?
Writing a Correct Program

```c
i = 1;
while(i < n){
    r = r * i;
    i++;
}
```

When n is 1, should return 1

When n is 2, should return 2
Writing a Correct Program

```c
i = 1;
while(i <= n){
    r = r * i;
    i++;
}
```

- When $n$ is 1, should return 1
- When $n$ is 2, should return 2
- When $n$ is 5, should return 120
- When $n$ is -1, should return?
Writing a Correct Program

When n is 1, should return 1
When n is 2, should return 2
When n is 5, should return 120

```java
i = 1;
while(i <= n){
    r = r * i;
    i++;
}
```
Writing a Proved-Correct Program

∀\( n \), \( f(n) = n! \)

\[
i = 1; \\
\text{while}(i < n)\{ \\
\quad r = r * i; \\
\quad i++; \\
\}\]

\[
x = v; \\
\text{rest} \\
\text{-- Semantics} \\
x: a \\
y: b \\
rest \\
x: v \\
y: b \\
\text{Semantics}
Writing a Proved-Correct Program

∀n, n ≥ 0 → f(n) = n!

i = 1;
while(i <= n){
  r = r * i;
  i++;
}
x = v;

Semantics

<table>
<thead>
<tr>
<th>x: a</th>
</tr>
</thead>
<tbody>
<tr>
<td>y: b</td>
</tr>
<tr>
<td>x: v</td>
</tr>
<tr>
<td>y: b</td>
</tr>
</tbody>
</table>

rest → rest
Writing a Proved-Correct Program

∀\(n, n \geq 0 \rightarrow f(n) = n!\)

```c
i = 1;
while(i <= n){
  r = r * i;
  i++;
}

x = v;
rest
```

**Semantics**

```
x: a
y: b
```

```
x: v
y: b
rest
```
The Coq Proof Assistant

• Common language for definitions and proofs
• Makes proofs easier
  — Keeps track of proof state
  — Easy lookup of definitions and already-known facts
  — Can write programming-style proof scripts
The Coq Proof Assistant

• Common language for definitions and proofs
• Makes proofs easier
• Makes proofs harder
  — Every step is checked by the system
  — Have to address every case
  — The computer takes everything literally
The Coq Proof Assistant

• Common language for definitions and proofs
• Makes proofs easier
• Makes proofs harder, so we know they’re correct proofs!
• Lots of existing projects: C compiler, operating system, Hoare logic, verification tools, web server, ...
Course Plan

• Introduction to Coq
• Basic logic and functional programming
• More advanced logic, mostly induction
• How to describe the behavior of programs

https://softwarefoundations.cis.upenn.edu/
Course Plan

• How to describe the behavior of programs
• Specifying and verifying programs
• Properties of programming languages
• How to verify C code

https://softwarefoundations.cis.upenn.edu/
Administrative Details

• Professor: William Mansky (mansky1@uic.edu)
• Meets MWF 2:00-2:50 PM in Lecture Center A2
• Office hours Monday 11:00-12:00, Wednesday 3:00-4:00 PM in SEO 1331, and by appointment
• Discussion board on Piazza
  — Both of these are great ways to get help with homework!
• Course website: https://www.cs.uic.edu/~mansky/teaching/cs494sf/sp19/
Class Format

• Each class will (usually) have a lecture and a lab section
• In lab, we’ll work on example problems and discuss practical issues
• Bring your laptop, with Coq installed and ready to use
• Participation in labs is 10% of the grade – you don’t have to finish all the problems, but you do have to make progress!
Assignments

• Mostly proof exercises from the textbooks
• Submit via Gradescope
• Due at 2 PM on the due date
• Automatic 2-day extension with a 20% penalty
• You can discuss strategy with other students, but don’t look at each other’s code!
• Cite your sources (websites, other students, stackoverflow, etc.)
Grading

- Class participation: 10%
- Assignments: 35%
- Midterms (2, in class): 30%
- Final: 25%
- Project (for grad students, opt-in for undergrads): 25%
Getting Started with Coq

• Available online at https://coq.inria.fr/
• You can download installers for Windows and Mac from the website, or use OPAM, the OCaml Package Manager
• Coq file extension is .v
• If it matters, we’ll use version 8.8.2

• Two main IDEs: CoqIde (ships with Coq) and ProofGeneral (Emacs mode)
• Both let you build interactive proof documents
To Do for Next Time

• Install Coq from https://coq.inria.fr/ or OPAM

• Download the textbooks (volumes 1 and 2) from https://softwarefoundations.cis.upenn.edu/, and compile them by running make

• Make sure you can run Coq on your machine, using either CoqIDE or ProofGeneral

• Next: basic definitions and proofs in Coq