CS 6390: Lecture 0
Course Overview

Prof. Bill Harris

QUIZ TIME

- This is NOT for a grade
- Take 10 - 15 mins
- Don’t know is a reasonable answer

Lecture overview

- Motivation
- Course topics
- Unofficial pre-requisites
- Grading

What you can get out of this course

1. Learn how to write correct programs

2. Learn how to design good languages

3. Learn “street-fighting” logic

How to write correct programs

- Writing correct programs is hard
- This course won’t give you a magic bullet that will make sure that you never write a bug again
- But it will:
    - Teach you how to write a program specification
    - Prove that a program satisfies the spec that you claim, on **all** inputs

How to design good languages

Ruby

```
>> a = b
NameError: undefined local variable or method 'b' for ma
in:Object
        from (irb):3
>> a = a
=> nil
>>
```

```
> [] + []

> [] + {}
[object Object]
> {} + []
0
> {} + {}
NaN
>
```

JavaScript

Language Design

- Best case: a joke; worst case: serious issues in program security, reliability
- In this course, you'll learn:
  - How to specify a language a prove properties of the language
  - How to show that a compiler/interpreter implements the language spec

Learn street-fighting logic

- Logic: been around in philosophy since antiquity
- Heavily studied in the early 1900’s in an attempt to unify all of mathematics
- A lot of mathematicians then were worried that it was too general to be useful

Learn street-fighting logic

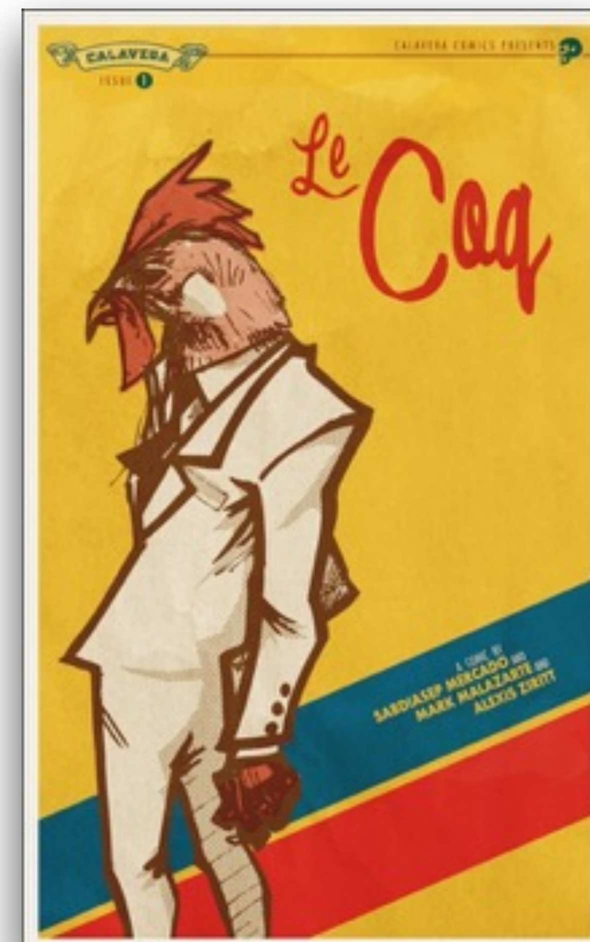
- Our secret weapon for reasoning about all programs, program states
- We'll use logic to:
  - Define program syntax
  - Define program semantics
  - Describe infinite sets of states, and how programs update them

Learn street-fighting logic

- Along the way, we'll gain a working appreciation for the **Curry-Howard correspondence**
    - Theorems = Types
    - Proofs of Theorem = Program of Type

Street-fighting logic

- Theorems will be stated and proven using the Coq proof assistant



- What Coq **does**:
  - Let’s you state a theorem precisely
  - Makes sure that your proof is correct
- What Coq **doesn’t** do:
  - Writes your proof for you
  - Let you grind out a proof without getting a good intuition

Coq scale

- In principle, Coq can be used to prove just about any mathematical theorem
- In practice, it’s a long way away from getting used to prove cutting-edge math
- Mostly gets used to prove theorems that encode system (compiler, OS) correctness
- Good systems have pretty boring correctness theorems **(demo)**

Course topics

- **Week 1:** Logic I
  - Equality
  - Induction
- **Week 2**: Functional Programming
  - Lists
  - Polymorphism

Course Topics

- **Weeks 3 - 4:** Logic II
  - Automation
- **Weeks 5 - 6**: Imperative programming
  - Define semantics
  - Prove program, language properties

Course Topics

- **Weeks 7 - 8:** program equivalence
  - Prove that two programs are equivalent
  - Prove that a transformation always produces equivalent programs
- **Weeks 9 - 10**: Hoare Logic
  - Specify pre-conditions and post-conditions
  - Prove that an imperative program satisfies them

Equivalence Example

```
WHILE (X!=1){
   HAVOC X
}
```

=

```
X:=1
```

Example program transformations

- Constant folding
- Algebraic identities

Hoare Logic Example

```
{ X = m }

WHILE (2 <= X) {
  X ::= X - 2;
}

{X = parity m }
```

Course Topics

- **Weeks 11 - 12**: Type systems
- **Weeks 13 - 16:** Lambda Calculi
  - Foundation for computing equivalent to Turing Machines
  - Core calculus of Lisp, Scheme, ML, Haskell, parts of Scala

Unofficial Prereqs

- Functional programming
  - Lisp, Scheme, ML, Haskell: core ideas
- Logic
  - Proof by induction
  - Formula validity (SAT)

Grading: Exercises

- Weekly exercises (**50%**). Each is one of:
  1. Given some lemma/theorem, prove it
  2. Given some informal property, formalize it, prove it
     - Coq pro: when you're right, you know it
     - Coq cons: if you don't are you left with nothing?
       - **No**: partial credit for intermediate lemmas, legible proofs

Exercises

3. Give an informal argument

   - We’ll grade on a published rubric

Grading

- Midterm, final (**40%**)

Grading

- In-class participation (**10%**)
  - Assigned reading for every lecture
  - Four - five times per lecture, I’ll pose a question
  - You answer, see a poll, discuss with your neighbor

Grading

- **Alternative**: project
  - Extend code-base of verified system (compiler, operating system, hypervisor, browser)
  - Reduced problem sets
  - Reduced weight for exams

Course Resources

- T-Square
- Piazza
- Text: **Software Foundations** (free)
- Coq (free)
- Proof-writing environments
    - CoqIDE: standalone development environment
    - Proof General: emacs mode

Some candid final notes

- This is an **experimental** course
  - First time we’ve offered a grad PL course in years
  - First time I’ve taught out of this book
  - Text is still actively edited
  - I’ll work with you to make sure the pace is right
- Be thoughtful of your friends on the waitlist
  (if they’re lucky)

Assignment for Wed

- Buy Clicker
- Install a proof environment
- Reply on the Piazza thread
  (what you like about your favorite language)
- Read **Software Foundations**, Ch. **Basics**, Sub-ch. **Numbers**
- Go to UROC

Assignment for Fri
(heads up)

- Work Coq proofs for some basic lemmas over natural numbers