Interactive Computer Theorem Proving

*Lecture 1: Why ICTP?*
About Me

- 4\textsuperscript{th} year CS PhD student in programming languages
- Started doing interactive computer theorem proving in Spring 2004, as part of the Open Verifier project
- Now it's the main focus of my research.
- Specifically, developing programming language tools with proofs of correctness
This Class

- A practical perspective on computer theorem proving
- Designed to be accessible to anyone who's taken a basic logic and discrete math class
- Experience with functional programming is a plus
  - Scheme/Lisp good, ML/Haskell better :-)


Administrivia

• Usually meet only on Thursdays
• One homework assignment a week during the first half of the course
  – Exercises using Coq (a proof assistant)
• For people taking the class for 3 units, a standard research project in a small group
  – Probably some application of interactive computer theorem proving
Administrivia II

- No required text, but the *Coq'Art* book is a useful reference
  - We have a few copies that we can loan out as needed
- This class probably won't satisfy any CS PhD breadth requirement, but see us if this is a problem for you.
What is a Proof?

- **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."
- **Proof by vigorous handwaving**
  - Works well in a classroom or seminar setting.
- **Proof by cumbersome notation**
  - Best done with access to at least four alphabets and special symbols.
- **Proof by exhaustion**
  - An issue or two of a journal devoted to your proof is useful.

[excerpt from a popular e-mail forwarding bonanza]
Classical Motivations

• Mathematicians and philosophers want to formalize their reasoning processes.

• Interest in formal methods driven by how difficult it is to be sure that a mathematical system corresponds to our intuitions.

• Want to come up with tiny but very expressive systems to study very carefully.
Don't Worry!

- This class is not about sitting around debating the metaphysics of “1 + 1 = 2.”
- We'll focus on a variety of practical applications of theorem proving technology.
- ...not that those philosophers didn't have some ideas that have turned out to be very practical. ;-)

Correctness is Nice

- Expensive mistakes
  - Pentium FDIV bug
  - Ariane rocket crash
  - etc.

- Programming language semantics
  - The POPLmark Challenge
The Age of "Security"

- The Internet isn't a friendly place anymore.
- "We want to make sure our software can't be exploited."
  - Verification of cryptographic protocols, etc.
- "We want to use software written by someone we don't trust."
  - Proof-carrying code
Software Engineering

• Developing programs and their correctness proofs simultaneously is an alternative to test-based development.
• The more intricate the system, the more likely it is that proof is more effective than testing.
• Exactly how to do this is a very active research topic today.
Goals for This Course

• Learn how to use the **Coq proof assistant** to:
  – Formalize most any kind of math
  – Formalize theory related to your research
  – Develop practical functional programs with total correctness proofs

• Learn exactly what it means for a proof to be rock solid, so that even a computer believes it.
The World of Computer Theorem Proving

First-Order Logic
- Untyped
  - Many systems...
- Automated
- Interactive
- ACL2

Higher-Order Logic
- Typed
  - Not so many systems...
- Functional Programming
- Ad-Hoc Proof Language
  - PVS
- Small Proof Language
  - Classical Logic
    - Isabelle
    - /HOL
  - Constructive Logic
    - Coq, NuPRL

Twelf

Logic Programming
• **Theorem**: There exist irrational numbers $a$ and $b$ such that $a^b$ is rational.

• If $\sqrt{2}^{\sqrt{2}}$ is rational, then we have the theorem with $a = b = \sqrt{2}$.

• If $\sqrt{2}^{\sqrt{2}}$ is irrational, then we have the theorem with $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$.

\[- \sqrt{2}^{\sqrt{2}} \cdot 2^{(\sqrt{2}^{\sqrt{2}})} = \sqrt{2}^2 = 2\]
A Constructive Proof

• **Theorem:** Every degree-one rational polynomial \( y = mx + b \) has a rational root if \( m \) is not 0.

• **Proof:** \(-b/m\) is the answer, because:
  \[
  -m(-b/m) + b = -b + b = 0
  \]

  rational root(rational m, rational b) {
    return -b / m;
  }

• **Precondition:** \( m \) is not 0.

• **Postcondition:** The return value is a root of \( y = mx + b \).
An Even Nicer Idea

• Theorem: Every Java program has an equivalent x86 machine language program.

• By choosing a suitable constructive logic, we guarantee that any proof of this theorem can be converted into a genuine Java compiler!

• By using a generic program extraction mechanism, we get the “free” theorem that our compiler preserves the semantics of programs.
  – ...which saves us a huge amount of testing.
Example: Alias Analysis

```c
int x, y;
int *p;
p = &y;
x = 1;
*p = 2;
return x;
```

Compiler Optimizer

```
return 1;
```

Empty intersection!

The path `x` only ever denotes elements of `{&x}`.
The path `y` only ever denotes elements of `{&y}`.
The path `*p` only ever denotes elements of `{&y}`.
Andersen's Analysis

\[ L: x = \text{new} \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \ Quadr.
Andersen in Coq

• A Coq implementation of Andersen's Analysis for this toy language, with a proof of total correctness

• Not quite so convoluted as you may be expecting from the slides on constructive logic, thanks to connections between proofs and functional programs that I haven't presented yet
But First...

How would you prove the correctness of Andersen's Analysis?

(if you had to convince someone who can only be convinced by a series of “obvious” steps)
Conclusion

• The full code of this example is available on the course web site.

• HW0 is posted
  – Install Coq and make sure you can run some simple examples through it.

• Next lecture: Revisiting freshman logic class
  – Natural deduction and interactive Coq proofs of theorems in propositional and first-order logic