

Lecture 1

CIS 500: SOFTWARE FOUNDATIONS

Steve Zdancewic

Spring 2016

Administrivia

- Instructor: Steve Zdancewic
Office hours: Tuesdays
Levine 511

- TAs:
 - Meyer Kizner
Office hours: TBA
 - Matthew Weaver
Office hours: TBA

Dr. Zdancewic will be at a research meeting 01/21 and 01/21

CLASS **CANCELLED**: 01/21
GUEST LECTURE: 01/26

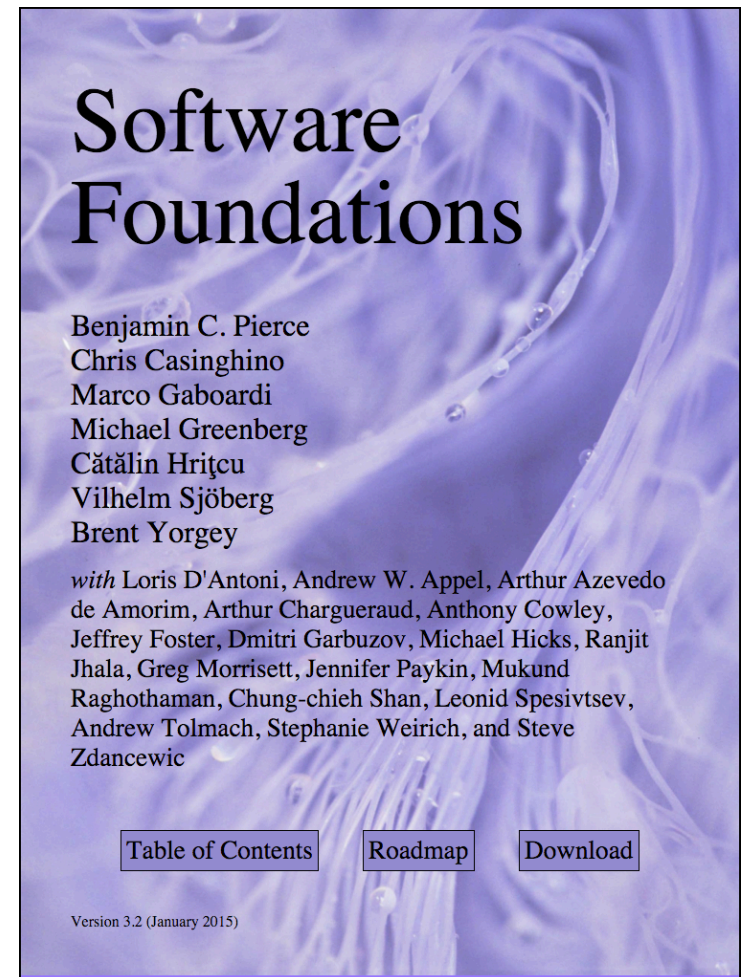
- Location: 303 Towne
- E-mail: cis500@seas.upenn.edu
- Web site: <http://www.seas.upenn.edu/~cis500>
- Canvas: <https://upenn.instructure.com>
- Piazza: <http://piazza.com/upenn/spring2016/cis500>

Resources

- Course textbook: *Software Foundations*
 - Electronic edition tailor-made for this class

Use the version available from the cis500 course web pages!

- Additional books:
 - *Types and Programming Languages* (Pierce, 2002 MIT Press)
 - *Interactive Theorem Proving and Program Development* (Bertot and Castéran, 2004 Springer)
 - *Certified Programming with Dependent Types* (Chlipala, electronic edition)



Course Policies

- Prerequisites:
 - Significant programming experience
 - Mathematical sophistication
 - Undergraduate functional programming or compilers class

Grading:

- 24% Homework ~12 weekly assignments
- 18% Midterm I (tentatively) Feb. 18th
- 18% Midterm 2 (tentatively) April 5th
- 36% Final
- 4% Class participation

⇒ Lecture attendance is crucial!

“Regular” and “Advanced” tracks (graded separately).

“Regular” vs. “Advanced” Tracks

- “Advanced” track:
 - More and harder exercises
 - More challenging exams.
 - It is a superset of the “regular” material.
- All students start in the advanced track by default.
- Students who wish to take CIS 500 for WPE I credit (Ph.D.) *must* take the advanced track.
- Students may switch from advanced to regular track at any time.
 - Notify the course staff in writing (by e-mail).
 - The change is *permanent* after the first midterm.
- Students wishing to switch (back) to the advanced track:
 - Must do so *before* the first midterm exam.
 - Must make up all the advanced exercises (or accept the grade penalty).
- Only students taking the advanced track are eligible for an A+.

Participation Policy

- Class attendance is mandatory.
- We will be using “clickers” for
 - in-class mini quizzes
 - in-class polls about course material
- TurningPoint clickers use will be your attendance record.
- For next time: *buy a clicker.*
- *Any TurningPoint RF clicker will work, see note on course website.*



Homework Policies

- Homework is to be done *individually*.
- Homework must be *submitted via Canvas*
- Homework that is late is subject to:
 - 25% penalty for 1 day late
 - 50% penalty for 2 days late
 - 75% penalty for 3 days late
- Homework is due at *8:00pm* on the due date (generally Thurs.).
- Advanced track students must complete (or try to complete) all non-optional exercises.
 - Missing “advanced” exercises will count against your score.
- Regular track students must complete (or try to complete) all non-optional exercises except those marked “advanced”.
 - Missing “advanced” exercises will not count against your score.
 - (But may help in your understanding of the material)

SOFTWARE FOUNDATIONS



LOGICAL FOUNDATIONS



A: How do we know something is true?

B: We test it out

A: But that isn't truth; testing can only give us evidence.
How do we know something is **true**?

B: We prove it

A: How do we know that we have a proof?

B: We need to define what it means to be a proof.

A proof is a logical sequence of arguments, starting from some initial assumptions

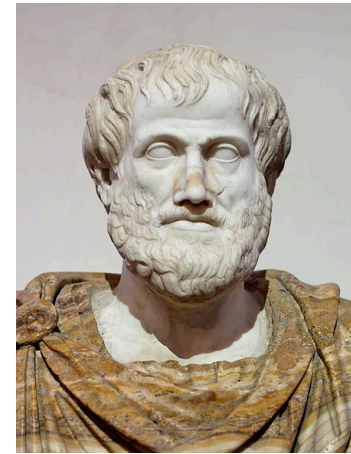
A: How do we know that we have a valid sequence of arguments? Can any list be a proof?

All humans are mortal

All Greeks are human

I am a Greek

B: No, no, no! We need to think about how we *think*....



Aristotle
384 – 322 BC



Euclid
~300 BC

First we need a language...

- **Gottlob Frege**: a German mathematician who started in geometry but became interested in logic and foundations of arithmetic.
- 1879 Published "*Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*" (Concept-Script: A Formal Language for Pure Thought Modeled on that of Arithmetic)
 - First rigorous treatment of functions and quantified variables
 - $\vdash A$, $\neg A$, $\forall x.F(x)$
 - First notation able to express arbitrarily complicated logical statements



Gottlob Frege
1848-1925



Formalization of Arithmetic

- 1884: *Die Grundlagen der Arithmetik* (The Foundations of Arithmetic)
- 1893: *Grundgesetze der Arithmetik* (Basic Laws of Arithmetic, Vol. 1)
- 1903: *Grundgesetze der Arithmetik* (Basic Laws of Arithmetic, Vol. 2)
- Frege's Goals:
 - isolate logical principles of inference
 - derive laws of arithmetic from first principles
 - set mathematics on a solid foundation of logic
- **David Hilbert**: a German recognized as one of the most influential mathematicians ever.
 - algebra, axiomatization of geometry, physics,...
 - 1900: published his "23 Problems"
 - Problem #2: Prove that the axioms of arithmetic are consistent



Hilbert
1943

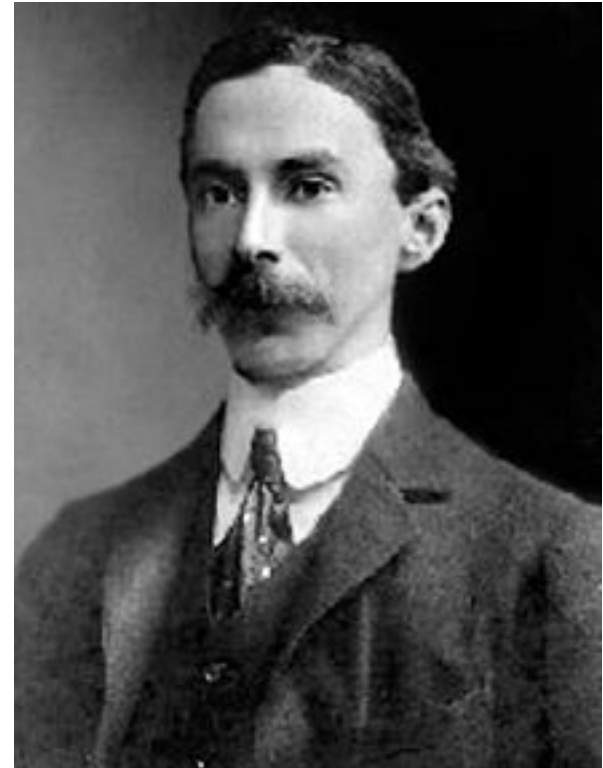
The plot thickens...

Just as Volume 2 was going to print in 1903,
Frege received a letter...

Bertrand Russell

- *Russell's paradox:*

1. Set comprehension notation:
 $\{ x \mid P(x) \}$ "The set of x such that $P(x)$ "
2. Let X be the set $\{ Y \mid Y \notin X \}$.
3. Ask the logical question:
Does $X \in X$ hold?
4. **Paradox!** If $X \in X$ then $X \notin X$.
If $X \notin X$ then $X \in X$.



Bertrand Russell
1872 - 1970

- Frege's language could derive Russell's paradox \Rightarrow it was *inconsistent*.
- Frege's logical system could derive anything. Oops(!!)

Addendum to Frege's 1903 Book

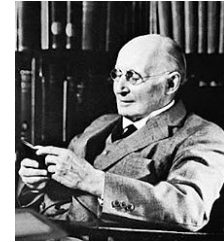
*“Hardly anything more unfortunate can befall a scientific writer than to have one of the **foundations** of his edifice shaken after the work is finished. This was the position I was placed in by a letter of Mr. Bertrand Russell, just when the printing of this volume was nearing its completion.”*

– Frege, 1903

Aftermath of Frege and Russell

- Frege came up with a fix, but it made his logic trivial...
- **1908**: Russell fixed the inconsistency of Frege's logic by developing a *theory of types*.
- **1910, 1912, 1913**, (revised **1927**):
Principia Mathematica (Whitehead & Russell)
 - Goal: axioms and rules from which *all* mathematical truths could be derived.
 - It was a bit unwieldy...

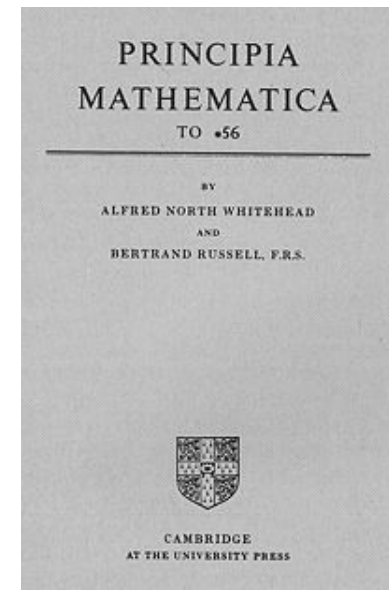
"From this proposition it will follow, when arithmetical addition has been defined, that $1+1=2$."
—Volume I, 1st edition, *page 379*



Whitehead



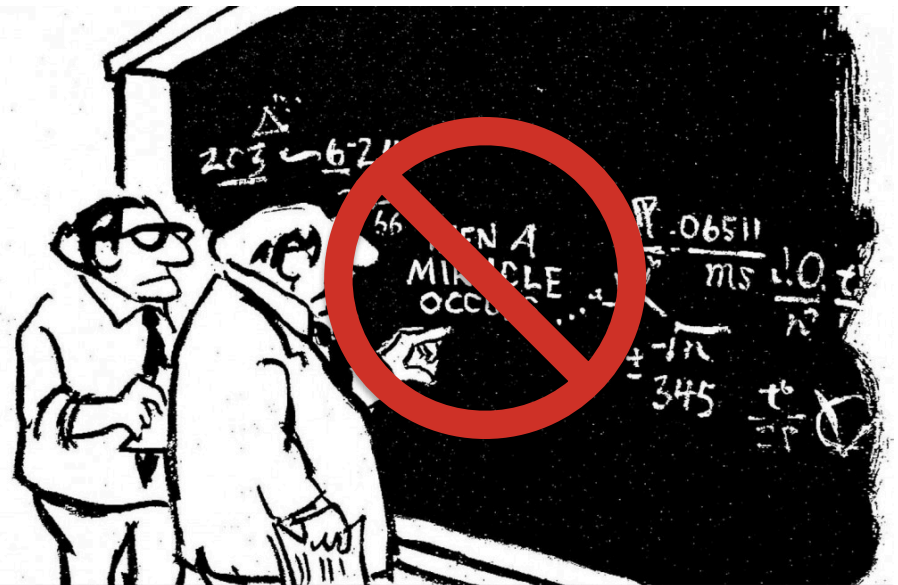
Russell



1920's: Hilbert's Program

A plan to secure the foundations of mathematics:

- Develop a *formal system* of all mathematics.
 - Mathematical statements should be written in a precise formal language
 - Mathematical proofs should proceed by well-specified rules
- Prove *completeness*
 - i.e. that all true mathematical statements can be proved
- Prove *consistency*
 - i.e. that no contradictory conclusions can be proved
- Prove *decidability*
 - i.e. there should be an algorithm for determining whether a given statement has a proof



Things were going well, following Russell & Whitehead, until...

Logic in the 1930s and 1940s

- **1931:** Kurt Gödel's first and second incompleteness theorems.
 - Demonstrated that any consistent formal theory capable of expressing arithmetic cannot be complete.
 - Write down: "This statement is not provable." as an arithmetic statement.
- **1936:** Genzen proves **consistency** of arithmetic.
- **1936:** Church introduces the **λ -calculus**.
- **1936:** Turing introduces Turing machines
 - Is there a decision procedure for arithmetic?
 - Answer: no it's undecidable
 - The famous "halting problem"
 - only in 1938 did Turing get his Ph.D.
- **1940:** Church introduces the **simple theory of types**



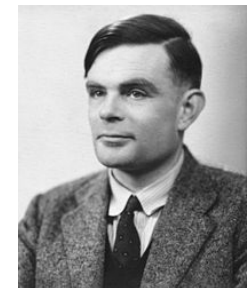
Kurt Gödel
1906 - 1978



Gerhard Gentzen
1909 - 1945



Alonzo Church
1903 - 1995



Alan Turing
1912 - 1954

Fast Forward...

- 1958 (Haskell Curry) and 1969 (William Howard) observe a remarkable correspondence:



Haskell Curry
1900 – 1982



William Howard
1926 –

types	~	propositions
programs	~	proofs
computation	~	simplification



N.G. de Bruijn
1918 - 2012

- 1967 – 1980's: N.G. de Bruijn runs Automath project
 - uses the Curry-Howard correspondence for computer-verified mathematics

- 1971: Jean-Yves Girard introduces System F
- 1972: Girard introduces $F\omega$
- 1972: Per Martin-Löf introduces intuitionistic type theory
- 1974: John Reynolds independently discovers System F

Basis for modern
type systems:
OCaml, Haskell,
Scala, Java, C#, ...

... to the Present

- 1984: Coquand and Huet first begin implementing a new theorem prover “Coq”
- 1985: Coquand introduces the **calculus of constructions**
 - combines features from intuitionistic type theory and $F\omega$
- 1989: Coquand and Paulin extend CoC to the **calculus of inductive constructions**
 - adds “inductive types” as a primitive
- 1992: Coq ported to Xavier Leroy’s **Caml**
- 1990’s: up to Coq version 6.2
- 2000-2010: Coq version 8.3
- 2012: Coq version 8.4pl6 ← CIS 500
- 2013: Coq receives ACM Software System Award



Thierry Coquand
1961 –



Gérard Huet
1947 –

Too many contributors
to mention here...

So much for foundations... what about software?

SOFTWARE FOUNDATIONS

Building Reliable Software

- Suppose you work at (or run) a software company.
- Suppose, like Frege, you've sunk 30+ person-years into developing the "next big thing":
 - Boeing Dreamliner2 flight controller
 - Autonomous vehicle control software for Nissan
 - Gene therapy DNA tailoring algorithms
 - Super-efficient green-energy power grid controller
- Suppose, like Frege, your company has invested a lot of material resources that are also at stake.
- How do you avoid getting a letter like the one from Russell?

Or, worse yet, *not* getting the letter to disastrous consequences?

Approaches to Software Reliability

- Social
 - Code reviews
 - Extreme/Pair programming
- Methodological
 - Design patterns
 - Test-driven development
 - Version control
 - Bug tracking
- Technological
 - “lint” tools, static analysis
 - Fuzzers, random testing
- Mathematical
 - Sound type systems
 - “Formal” verification



Less “formal”: Techniques may miss problems in programs

This isn’t a tradeoff... all of these methods should be used.

Even the most “formal” can still have holes:

- Did you prove the right thing?
- Do your assumptions match reality?
- Knuth. “Beware of bugs in the above code; I have only proved it correct, not tried it.”

More “formal”: eliminate *with certainty* as many problems as possible.

Five Interwoven Threads

1. basic tools from **logic** for making and justifying precise claims about programs
2. the use of **proof assistants** to construct rigorous, machine checkable, logical arguments
3. the idea of **functional programming**, both as a method of programming and as a bridge between programming and logic
4. techniques for formal **verification** of properties of specific programs
5. the use of **type systems** for establishing well-behavedness guarantees for all programs in a given language

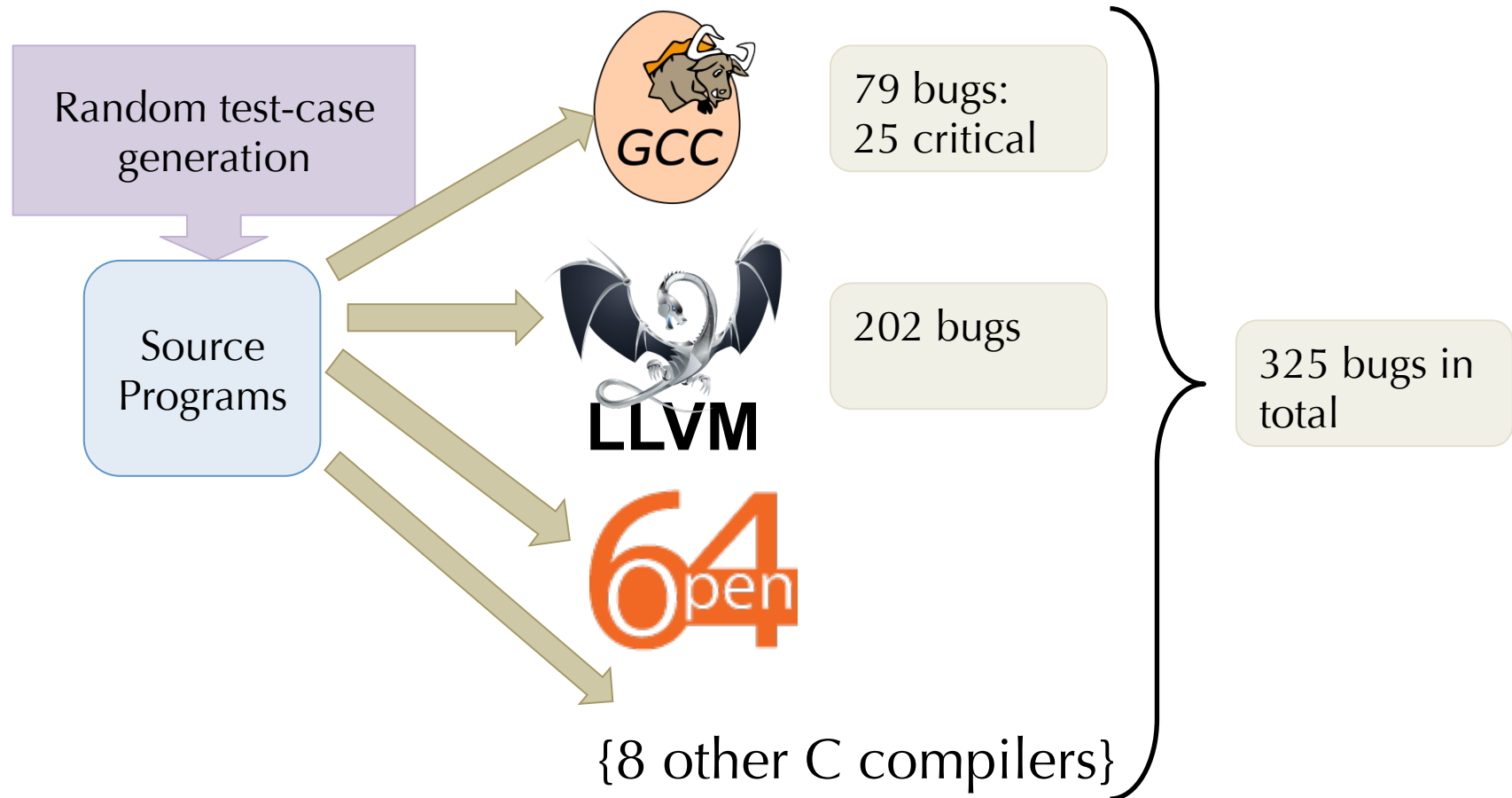
Can it Scale?

- Use of theorem proving to verify “real” software is still considered to be the bleeding edge of PL research.
- **CompCert** – fully verified C compiler
Leroy, INRIA
- **Vellvm** – formalized LLVM IR
Zdancewic, Penn
- **Ynot** – verified DBMS, web services
Morrisett, Harvard
- **Verified Software Toolchain**
Appel, Princeton
- **Bedrock** – web programming, packet filters
Chlipala, MIT
- **CertiKOS** – certified OS kernel
Shao & Ford, Yale



Does it work?

Finding and Understanding Bugs in C Compilers [Yang et al. PLDI 2011]



Verified Compiler: CompCert [Leroy et al.]
<10 bugs found in *unverified* front-end component

Regehr's Group Concludes

The striking thing about our CompCert results is that the *middle-end bugs* we found in all other compilers are *absent*. As of early 2011, the under-development version of *CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors*. This is not for lack of trying: we have devoted about six CPU-years to the task. *The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users.*

(emphasis mine)


The logo for 'deep spec' is a purple square with a white vertical line on the left side. The word 'deep' is written in white lowercase letters above the word 'spec', which is also in white lowercase letters.

the science of deep specification

- National Science Foundation "Expedition" Project
 - \$10M over five years
 - Penn: Zdancewic / Weirich / Pierce
 - Princeton: Appel
 - Yale: Shao
 - MIT: Chlipala
- Many ways to get involved (especially after CIS 500!)
- See www.deepspec.org



Why CIS 500?

- **Foundations**
 - Functional programming
 - Constructive logic
 - Logical foundations
 - Proof techniques for inductive definitions
- **Semantics**
 - Operational semantics
 - Modeling imperative “While” programs
 - Hoare logic for reasoning about program correctness
- **Type Systems**
 - Simply typed λ -calculus
 - Type safety
 - Subtyping
 - Dependently-typed programming
- **Coq interactive theorem prover**
 - turns doing proofs & logic into programming  fun!

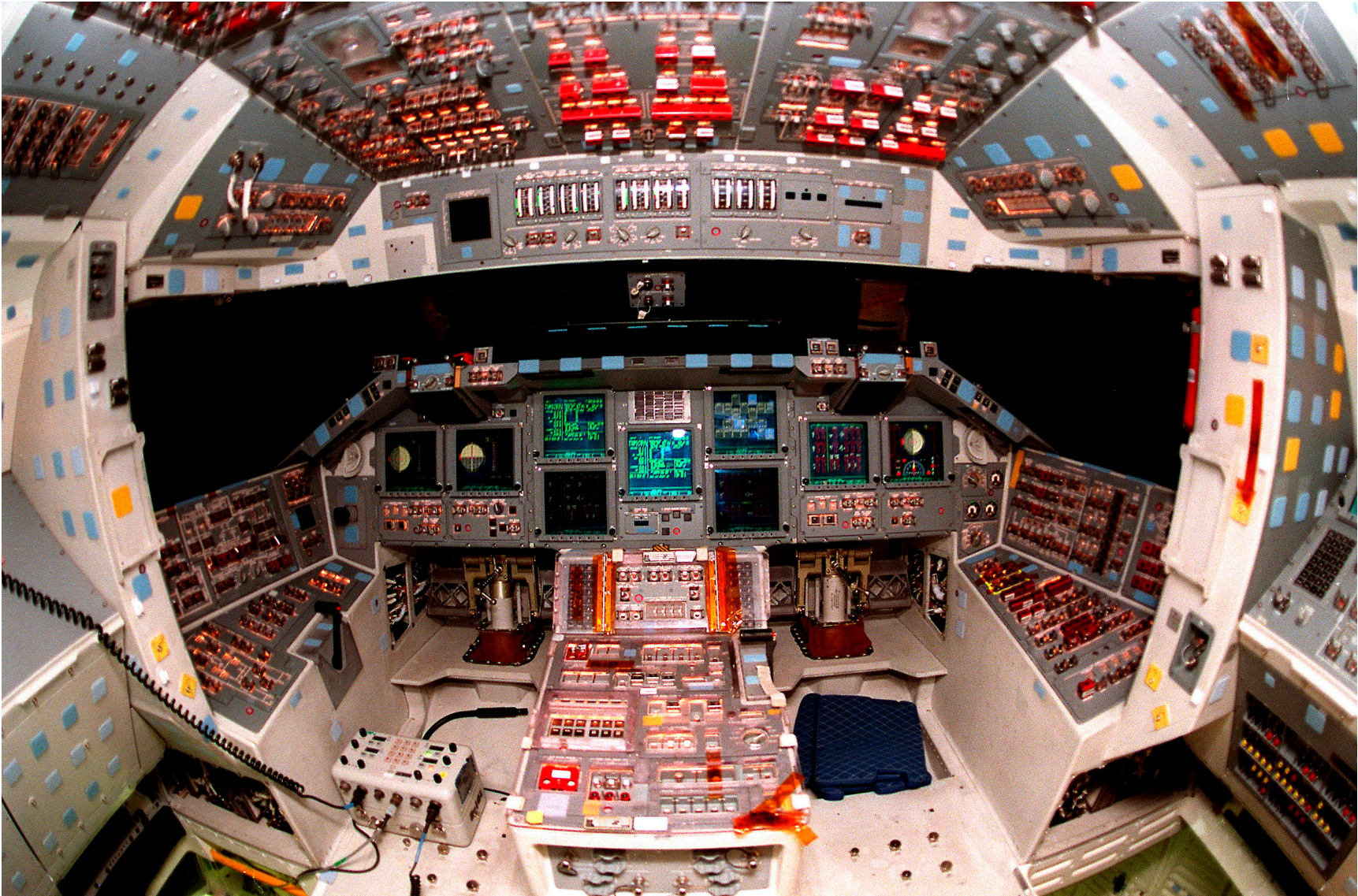
COQ

Coq in CIS 500

- We'll use Coq version 8.4pl6
 - Available on CETS systems
 - Easy to install on your own machine
- See the web pages at: coq.inria.fr
- Two different user interfaces
 - CoqIDE – a standalone GUI / editor
 - ProofGeneral – an Emacs-based editing environment
- Course web pages have more information.



Coq's Full System



Subset Used in CIS 500



To start.



By the end of the semester.

Getting acquainted with Coq.

BASICS.V

CIS 500: TODO

- Soon:
 - Register for Piazza
 - Try to log in to Canvas
 - Reading: Preface and Basics
- Before next time:
 - Install Coq v. 8.4pl6
 - Obtain a clicker
- HW1: Finish Basics.v
 - Due: Thursday, January 21st at 8:00pm
 - Available on the web pages
 - Complete all non-optional exercises
 - There are no “advanced” problems for this HW
 - Submit to Canvas