Lecture 1

(VIRTUAL) CIS 500: SOFTWARE FOUNDATIONS

Benjamin Pierce

Fall 2020

SOFTWARE FOUNDATIONS

How do we **build** software?

^

that works

^

(and be convinced that it does)

Critical Software

Individual programs

- Operating systems
- Network stacks
- Crypto
- Medical devices
- Flight control systems
- Power plants
- Home security
- …

Programming languages

- Compilers
- Static type system
- Data abstraction and modularity features
- Security controls

Logic

+ Reasoning about individual programs

+ Reasoning about whole programming languages

SOFTWARE FOUNDATIONS

LOGICAL FOUNDATIONS

Q: How do we know something is true?

A: We prove it

Q: How do we know that we have a *proof*?

A: We need to know what it means for something to be a proof.
First cut: A proof is a “logical” sequence of arguments, starting from some initial assumptions

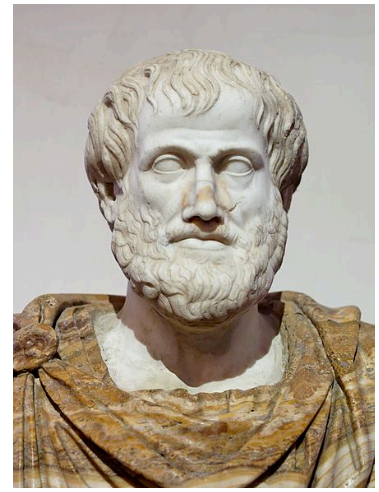
Q: How do we agree on what is a *valid* sequence of arguments? Can any sequence be a proof? E.g.

All humans are mortal

All Greeks are human

Therefore I am a Greek!

A: No, no, no! We need to think harder about valid ways of reasoning...



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

BEGRIFFSSCHRIFT,

EINE DER ARITHMETISCHEN NACHGEBILDETE

FORMELSPRACHE

DES REINEN DENKENS.

VON

Dr. GOTTLOB FREGE.

HALLE a/S.

VERLAG VON LOUIS NEBERT.

1879.

Images in this & following slides taken from Wikipedia.

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

The plot thickens…

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

Addendum to Frege's 1903 Book

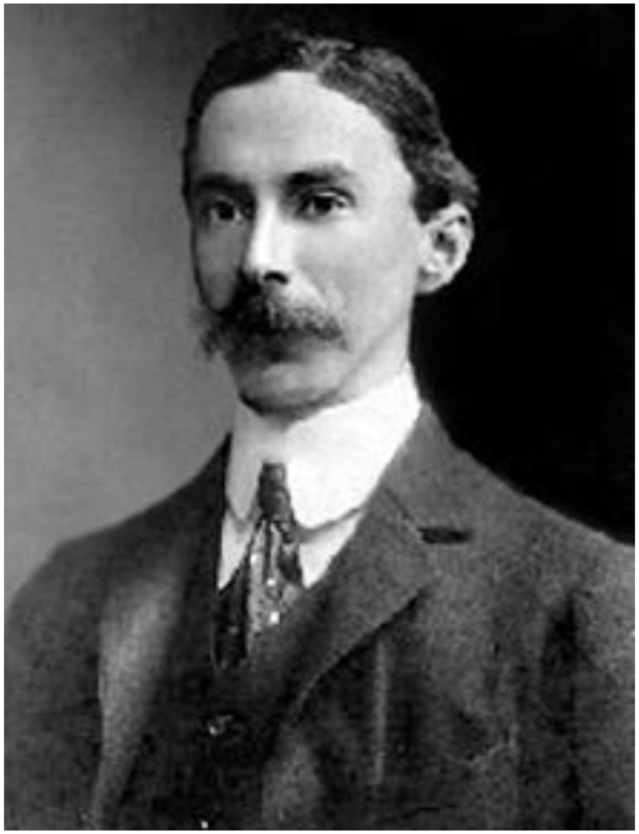
"Hardly anything more unfortunate can befall a scientific writer than to have one of the foundations of his edifice shaken after the work is finished. This was the position I was placed in by a letter of Mr. Bertrand Russell, just when the printing of this volume was nearing its completion."

– Frege, 1903

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 (of sets) $\{ Y \mid Y \notin Y \}$.
>
> 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 ⇒ it was *inconsistent*.
- Frege's logical system could derive anything. (Oops!)

Aftermath of Frege and Russell

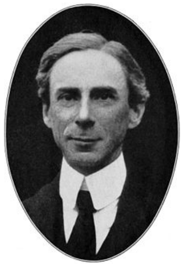
- Frege came up with a fix… but it made his logic trivial :-(

- 1908: Russell fixed the inconsistency of Frege's logic by developing a *theory of types*.

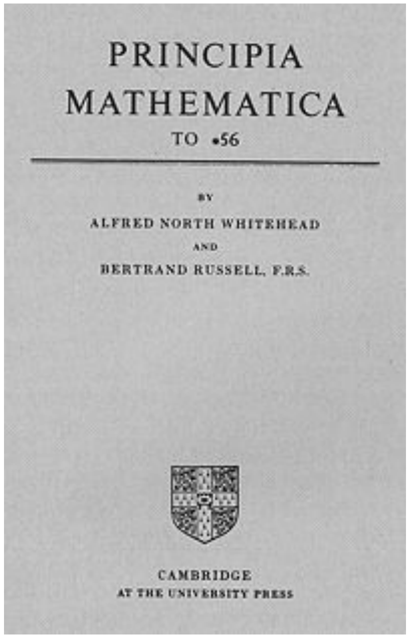
- 1910, 1912, 1913, (revised 1927):
  Principia Mathematica (Whitehead & Russell)
  - Goal: axioms and rules from which *all* mathematical truths could be derived.
  - It was a bit unwieldy…

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

Whitehead



Russell



PRINCIPIA
MATHEMATICA
TO ∗56

BY
ALFRED NORTH WHITEHEAD
AND
BERTRAND RUSSELL, F.R.S.

CAMBRIDGE
AT THE UNIVERSITY PRESS

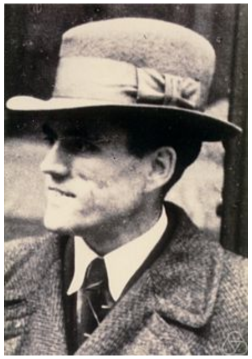
Logic in the 1930s and 1940s

- 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"
    - N.b.: 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…

- Two logicians in 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

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

- 1971: Jean-Yves Girard introduces System F
- 1972: Girard introduces Fω
- 1972: Per Marin-Löf introduces intuitionistic type theory
- 1974: John Reynolds independently discovers System F

… 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 Fω
- 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 OCaml
- 1990's: up to Coq version 6.2
- 2000-2015: up to Coq version 8.4
- 2020: Coq version 8.12 ← CIS 500

Too many contributors to list here…

- 2013: Coq receives ACM Software System Award

Thiery Coquand
1961 –

Gérard Huet
1947 –

So much for foundations… what about the “software” part?

PROGRAMMING (LANGUAGE) 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,
with disastrous consequences down the road?

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": Lightweight, inexpensive techniques (that may miss problems)

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

Even the most "formal" argument 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.

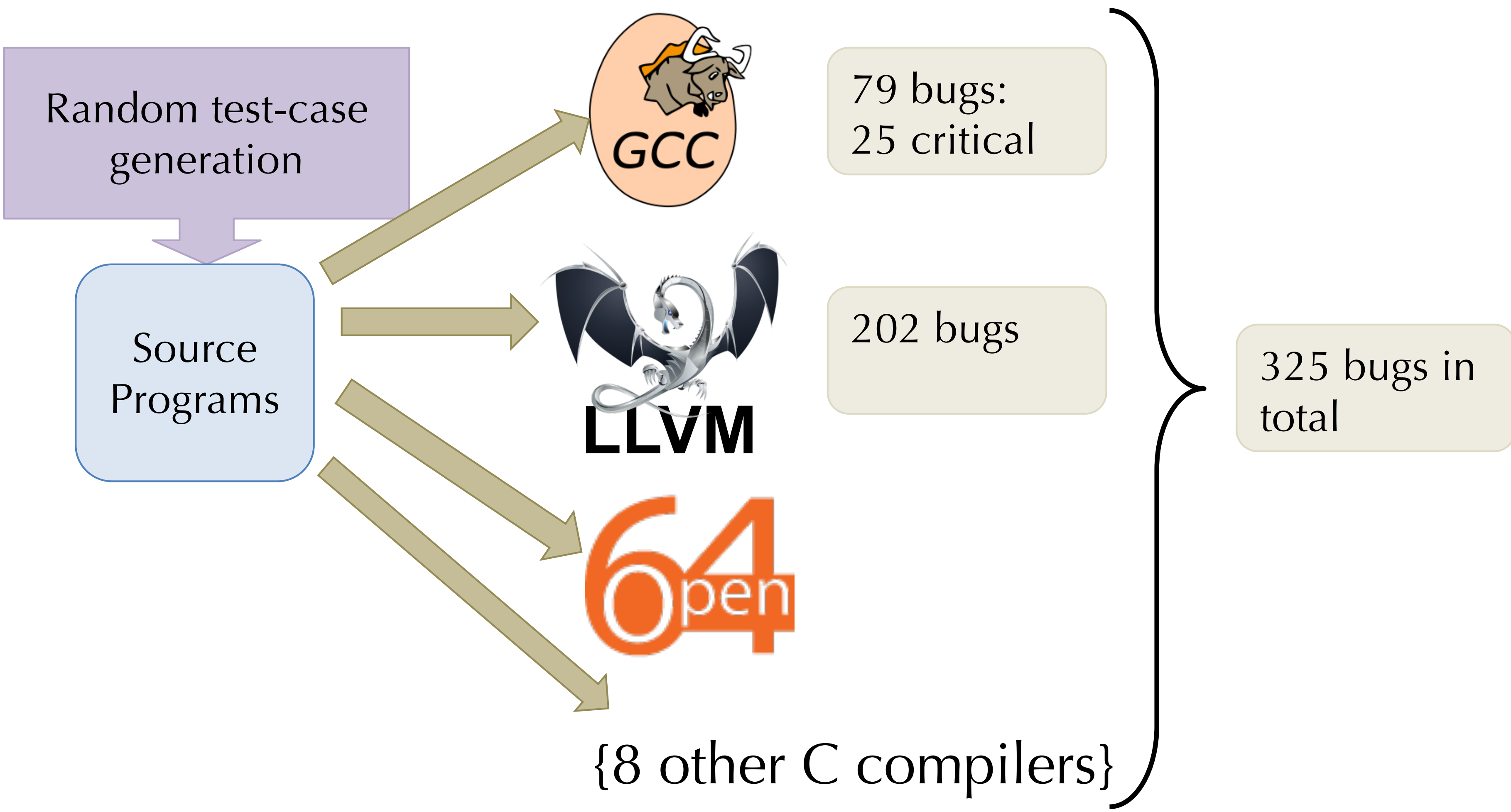
Can formal methods scale?

Use of formal methods to verify full-scale software systems is a hot research topic!

- 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 (at the time *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.*

deep spec

the science of deep specification

- National Science Foundation "Expedition" Project
  - $10M over five years
  - Penn: Pierce / Weirich / Zdancewic
  - Princeton: Appel
  - Yale: Shao
  - MIT: Chlipala

- Many ways to get involved (especially after CIS 500!)
- See www.deepspec.org

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!

COURSE MECHANICS

Administrivia

- Instructor: Benjamin Pierce
  Office hours: See web page (currently Mondays 1:30-3:30)
  Location TBA

- TAs:
  - Lucas Silver
  - Irene Yoon

- Location: Zoom
- E-mail: cis500@seas.upenn.edu (goes to all course staff)
- Web site: http://www.seas.upenn.edu/~cis500
- Canvas: https://upenn.instructure.com
- Piazza: http://piazza.com/upenn/fall2020/cis500

Resources

- Course textbook: *Software Foundations, volumes 1 and 2*
  - Electronic edition tailor-made for this class

Use the version available from the cis500 course web pages!!

(A new version of each chapter will generally go live just before class. :-)

- Additional resources:
  - *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)

How to CIS500

Live

- Live lectures will be as interactive as possible!
- Keep your video on
- Ask lots of questions
- Focus on the class instead of multitasking

Async

- Every lecture will be recorded
- Should be available on Canvas a few hours later
- Feel free to use them (and the textbook) instead of attending live if that works better for you

Course Policies

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

Grading:

- 25% Homework (~12 weekly assignments)
- 20% Midterm I (in class, early October)
- 20% Midterm 2 (in class, early November)
- 35% Final (date TBA)

“Regular” vs. “Advanced” Tracks

- “Advanced” track:
  - More and harder exercises
  - More challenging exams
  - Covering a superset of the “regular” material

- Everybody starts in the advanced track by default.
- Students who wish to take CIS 500 for both course credit and WPE I credit (Ph.D.) *must* follow 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+.
- “Regular” and “Advanced” tracks are curved separately

Class Participation

- We will be using Poll Everywhere, an online polling platform, for
  - in-class mini quizzes
  - real-time "polls" during lectures

- For next time: *download the Poll Everywhere app for your smartphone.*

Homework Policies

- Homework must be done *individually*
- Homework must be *submitted via Canvas*
- Homework that is late is subject to:
  - 25% penalty for 1 day late (up to 24 hours after deadline)
  - 50% penalty for 2 days late
  - 75% penalty for 3 days late

- Homework is due at *11:30am* on the due date
- Advanced track students must complete (or attempt) all non-optional exercises *including* those marked "advanced".
  - Missing "advanced" exercises will count against your score.
- Regular track students must complete (or attempt) all non-optional exercises *except* those marked "advanced".
  - Missing "advanced" exercises will *not* count against your score.
  - But you are welcome to try them!

WPE-I Policy

- If you wish to take CIS500 for WPE-I (Written Preliminary Exam, part I) credit toward a CIS PhD degree, you have two choices:
  - **Final exam only** option: WPE-I credit only (no need to be registered for the course). Passing score for WPE-I credit is determined by the CIS500 instructors (Pierce, Weirich, Zdancewic). Historically, this has been around a B+ grade on the exam.
  - **Full course participation** option: Must be registered for the course. WPE-I credit awarded for a weighted average grade of B+ on homework and all three exams.
    - You can take the course P/F and also receive WPE-I credit (following the same criteria)

TODO (for you)

- Before next class:
  - Register for Piazza (if you are not already registered)
  - Try to log in to Canvas
  - Install Coq (version 8.12)
  - Download Poll Everywhere app on your phone
  - Start reading: Preface and Basics

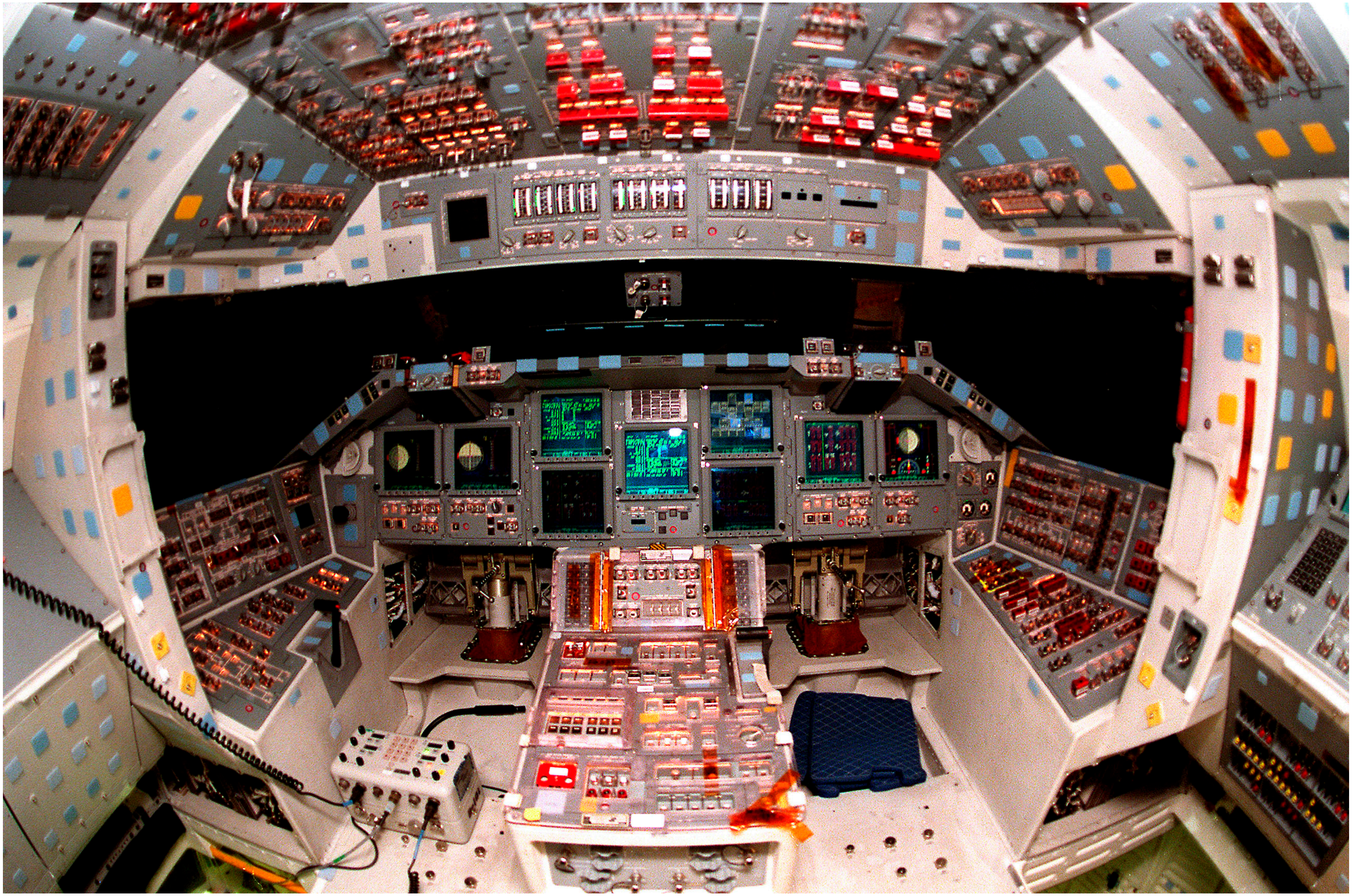
- HW1: Exercises in Basics.v
  - Due: Tuesday, September 8th at 11:30AM
  - Available from course web page
  - Complete all non-optional exercises
    - There are no "advanced" problems for this HW
  - Submit via Canvas

COQ

Coq in CIS 500

- We'll use Coq version 8.12
  - 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 Capabilities



Subset Used in CIS 500

To start.

By the end of the semester.

Getting acquainted with Coq…

BASICS.V