CS 565: Programming Languages

Fall 2017

Tu, Th: 12 - 1:15

Haas G066

Administrivia

- Course Web Page:
  - http://www.cs.purdue.edu/homes/suresh/565-Fall2017
- Office Hours:
  - Tu, Th: 4 - 5pm
  - By appointment
- Main text:
  - Software Foundations: hypertext version available from the course web page

Course Work

- Lectures
- Homeworks
- Problem Sets
- Midterm
- Cumulative final which will also serve as the qualifying exam

Prerequistes

- Programming experience/maturity
  - exposure to various language constructs
    - Java, ML, Lisp, Prolog, C
  - Undergraduate compilers and/or PL class
    - CS 352 and/or CS 456 or equivalent
- Mathematical maturity
  - familiarity with first-order logic, set theory, graph theory, induction
- Most important:
  - Intellectual curiosity and creativity

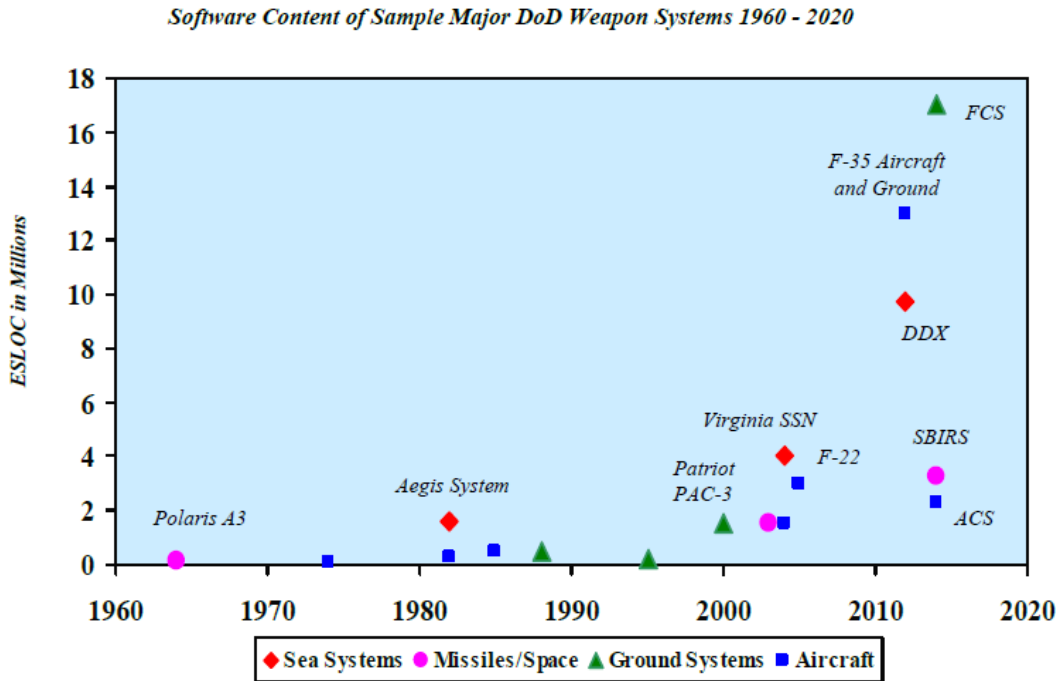
Resources

- Web page for text:
  - http://softwarefoundations.cis.upenn.edu
  - Certified Programming with Dependent Types
  - Types and Programming Languages
- Coq resources:
  - http://coq.inria.fr
- Proceedings of conferences
  - POPL, PLDI, ICFP, ...

Background

- Our main goal is to find ways to describe the behavior of programs precisely and concisely
- Motivation
  - Software complexity
  - Security
  - Productivity and understanding

Complexity

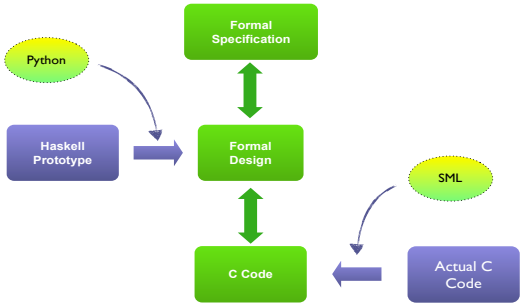


Rise of Software-Intensive DoD Systems:

- 35% of all Government systems use open source software
  [Open Source Survey, 2013]
- F-35 has *14x* more code (24MLoC) than F-22 and *175x* more code than F-16; software remain number one technical risk
  [AT Kearney Report, 2013], [BreakingDefense.com, March 2014]
- For weapons systems, 90% of its functions are performed by software
  [NAVAIR and U.S. Army Communications-Electronics Life Cycle Management Command (CECOM), 2010]

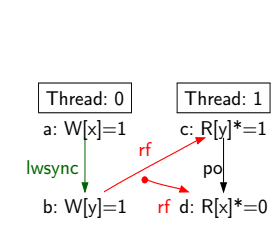
Complexity

Specifications may be complex

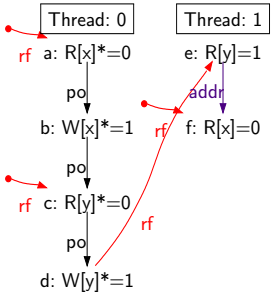


Sel4 architecture: Specification defined across multiple abstraction layers which include various untrusted translation phases.

Specifications may be intensional



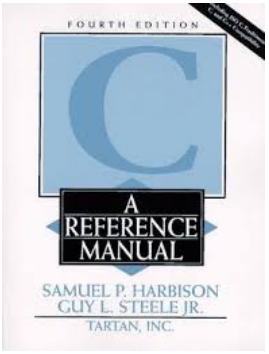
Test MP+lwsync+porr: Forbidden



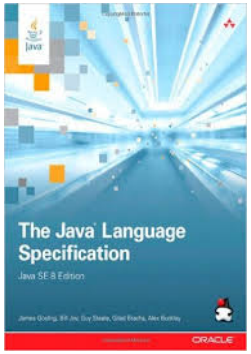
Test MP+poaa+addr: Allowed

Power relaxed-memory model: Allowed program behaviors depend on visibility and ordering guarantees of underlying processor.

Specifications may be tedious



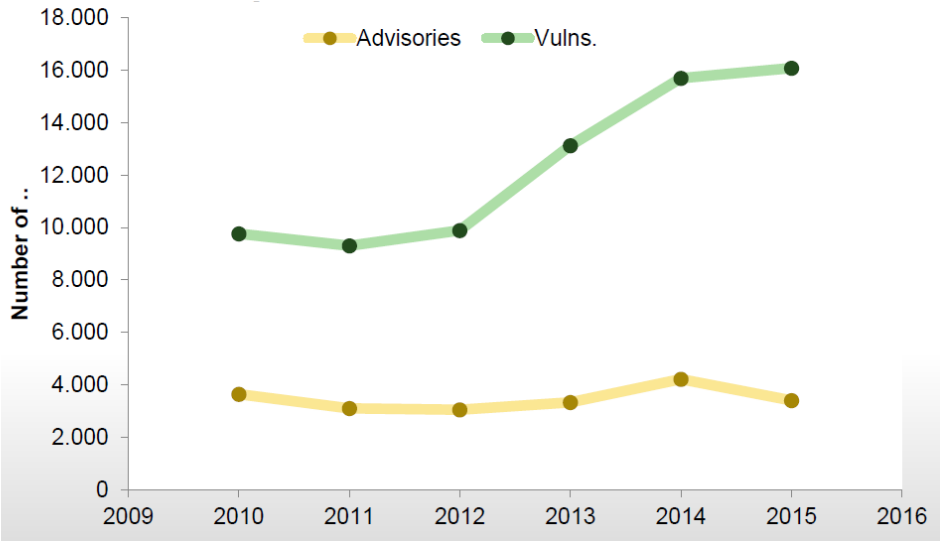
592 pages
1163 semantic rules in K



792 pages
1074 semantic rules in K

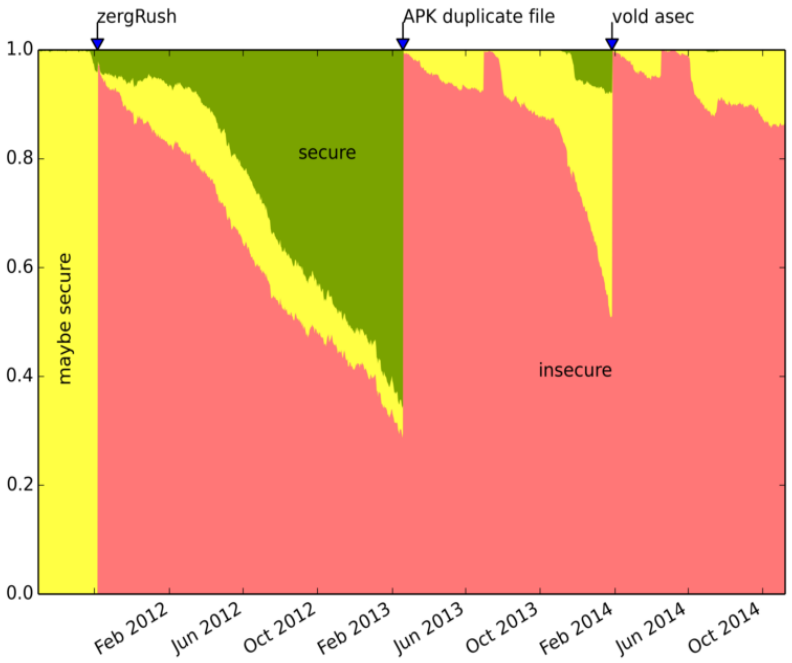
545 pages
1370 semantic rules in K

Security



Flexera Software Vulnerability Review, March 16, 2016

Proportion of Android devices running insecure, maybe secure, and secure versions of Android over time



Vulnerability History

- 2,484 applications from 263 vendors
- 40% Microsoft applications and 60% non-Microsoft applications in Windows operating systems

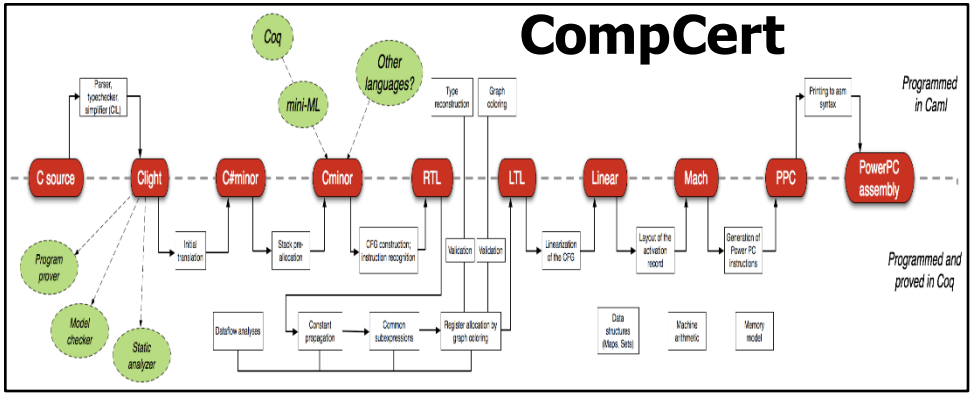
Trustworthiness through First Principles

VeLLVM

838 LoC; 50K LoP

HACMS quadcopter

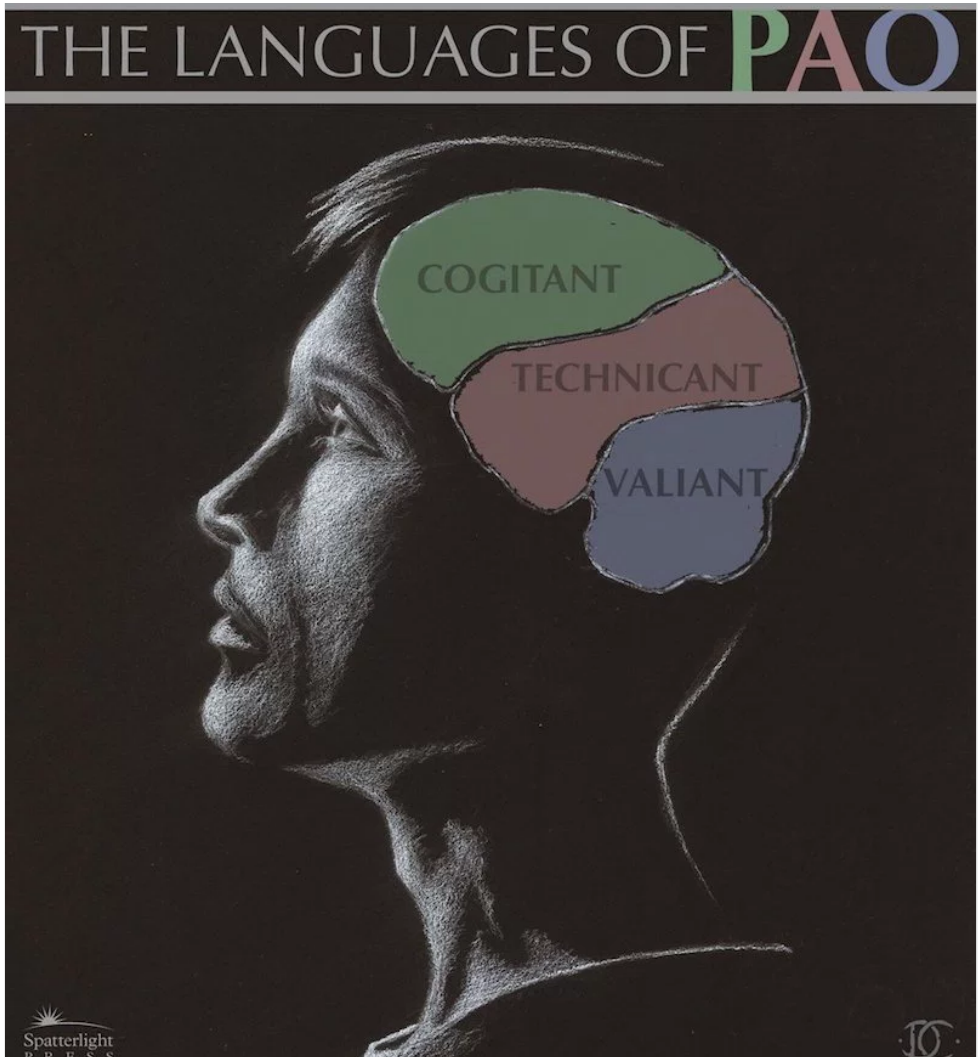
CompCert



8K LoC; 50K LoP

9K LoC; 400K LoP

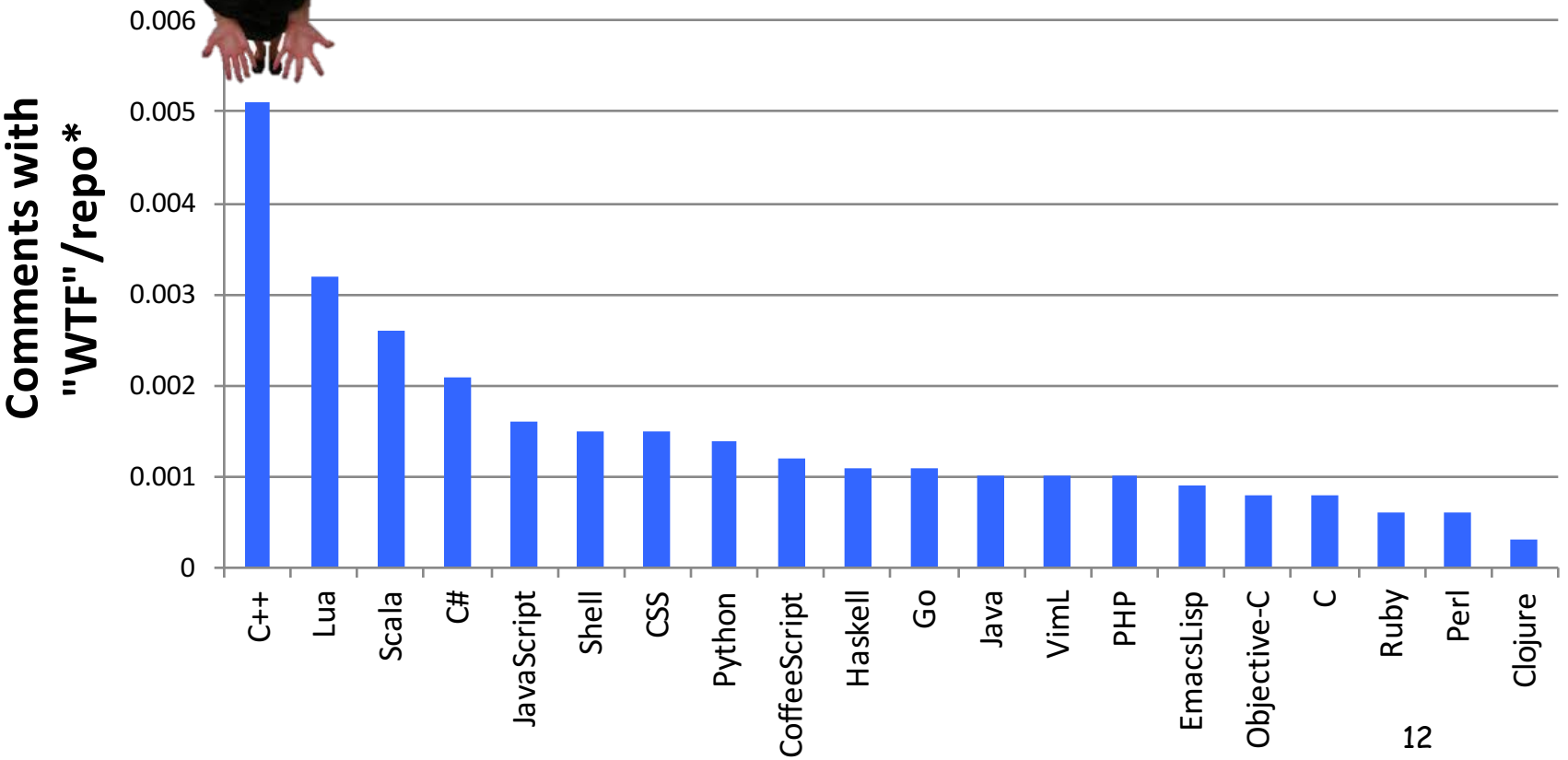
Productivity and Understanding



Everything is terrible.

Most software, even critical system software,
is insecure Swiss cheese
held together with duct tape, bubble wrap,
and bobby pins.

Anecdotal evidence ….

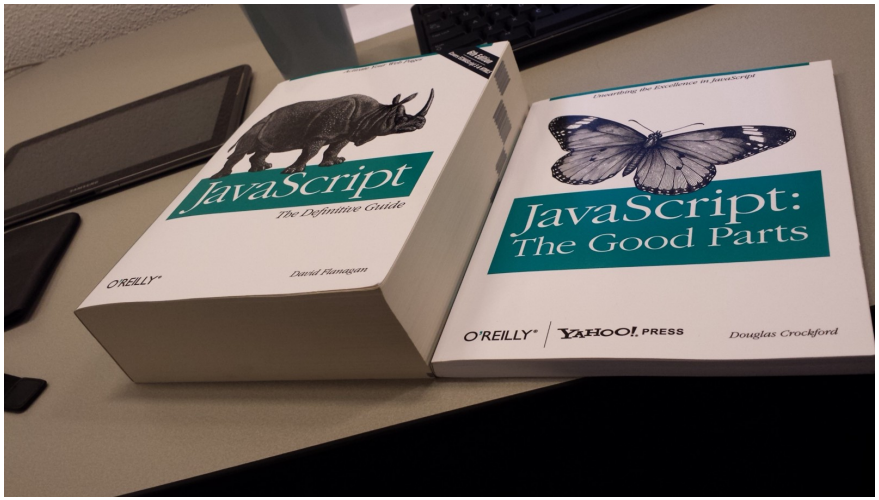


Javascript strangeness …

```
$ jsc
> [] + []

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

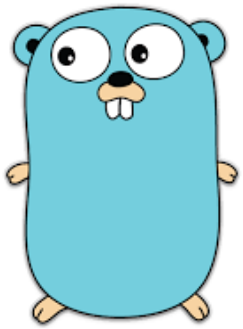
```
$ jsc
> Array(14)
,,,,,,,,,,,,,,
> Array(14).join("foo")
foofoofoofoofoofoofoofoofoofoofoofoofoo
> Array(14).join("foo" + 1)

> Array(14).join("foo" - 1) + "Batman!"
NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman!
```

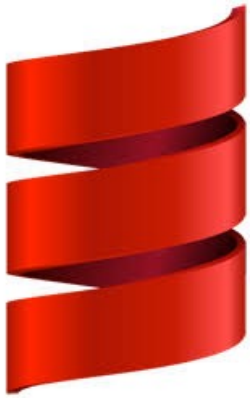
JavaScript is a Dysfunctional Programming Language



Design Choices

THE C PROGRAMMING LANGUAGE





Approaches

Testing

PLT Redex: DSL for specifying, debugging, and testing operational semantics

Quickcheck: Specification-driven formulation of properties that can be checked using random testing

Csmith: Random generator of C programs that conform to C99 standard for stress-testing compilers, analyses, etc.

CUTE: Unit-testing of C programs with pointer arguments by combining symbolic and concrete executions

Korat: Constraint-based generation of complex test inputs for Java programs, focusing on data structures and invariants

Symbolic Execution

KLEE: Symbolic execution engine to generate high-coverage test cases

S2E: Scalable path-sensitive platform

Static Program Analysis

CFA: Whole-program control-flow analysis that computes the set of procedures that can be invoked at a call-site

ASTREE: Abstract interpretation of real-time embedded software designed to prove absence of runtime errors by overapproximation of program behavior

TVLA: Flow-sensitive shape analysis of dynamically-allocated imperative data structures

Bddbdddb: Context- and field-sensitive analysis applied to Java that translates analysis rules expressed in Datalog to BDD representation

Saturn: Scalable and modular summary-driven bit-level constraint-based analysis framework

Coverity: Unsound scalable analyses used to check correctness of C, C++, and Java programs.

Dynamic Program Analysis

Contracts : Assertions checked at runtime with blame

Daikon: Likely pre- and post-condition invariant detection over propositional terms, based on program instrumentation

Valgrind: Instrument binary programs to track memory access violations and data races using dynamic recompilation

Fasttrack: Lightweight data race detector that uses vector clocks and a dynamically constructed happens-before relation

Model Checking

CVC, SLAM, Blast, Spin, Java PathFinder

CHESS: Bounded model-checking for unit-testing of shared-memory concurrent programs

TLA: Temporal logic of actions for specifying and checking concurrent systems

Logics and Types

Jstor, Space Invader, Smallfoot: Separation-logic based tools for verifying expressive shape properties of dynamic data structures and heaps

ESC: Extended static checking that combines type checking with theorem proving

Coq. Agda, Isabelle, ACL2, NuPRL: Mechanized proof assistants

Ynot: Hoare Type Theory

Rely-Guarantee Reasoning: Modular verification of shared-memory concurrency

Liquid Type Inference: Discovery of expressive refinement properties in Haskell, ML, and C

Hybrid Type Checking and Soft Typing

Session Types: type systems for expressing communication protocols

Goals

- A more sophisticated appreciation of programs, their structure, and the field as a whole
  - Viewing programs as rich, formal, mathematical objects, not mere syntax
  - Define and prove rigorous claims about a program's meaning and behavior
  - Develop sound intuitions to better judge language properties
- Develop tools to be better programmers, software designers, computer scientists, and thinkers

Non-goals

- An introduction to advanced programming techniques
- No detailed discussion of (low-level) machine implementations
  - The course will not be motivated from the perspective of a compiler writer
  - But, impact of design decisions on implementation tractability will be considered when appropriate
- A survey of different languages

Topics

- Part I (Tools): Proof assistants and functional programming
- Part II (Foundations): Program logic and reasoning principles
- Part III (Semantics and specifications): operational, axiomatic
- Part IV (Type Systems): λ-calculus, records, references, polymorphism, subtyping, inheritance, ...

Homework

- Install Coq
  - version 8.5pl
- Install either
  - Proof General
    - http://proofgeneral.inf.ed.ac.uk
  - Coq ide
    - available as part of the Coq distribution
- Answer "Admits" in Basics.v