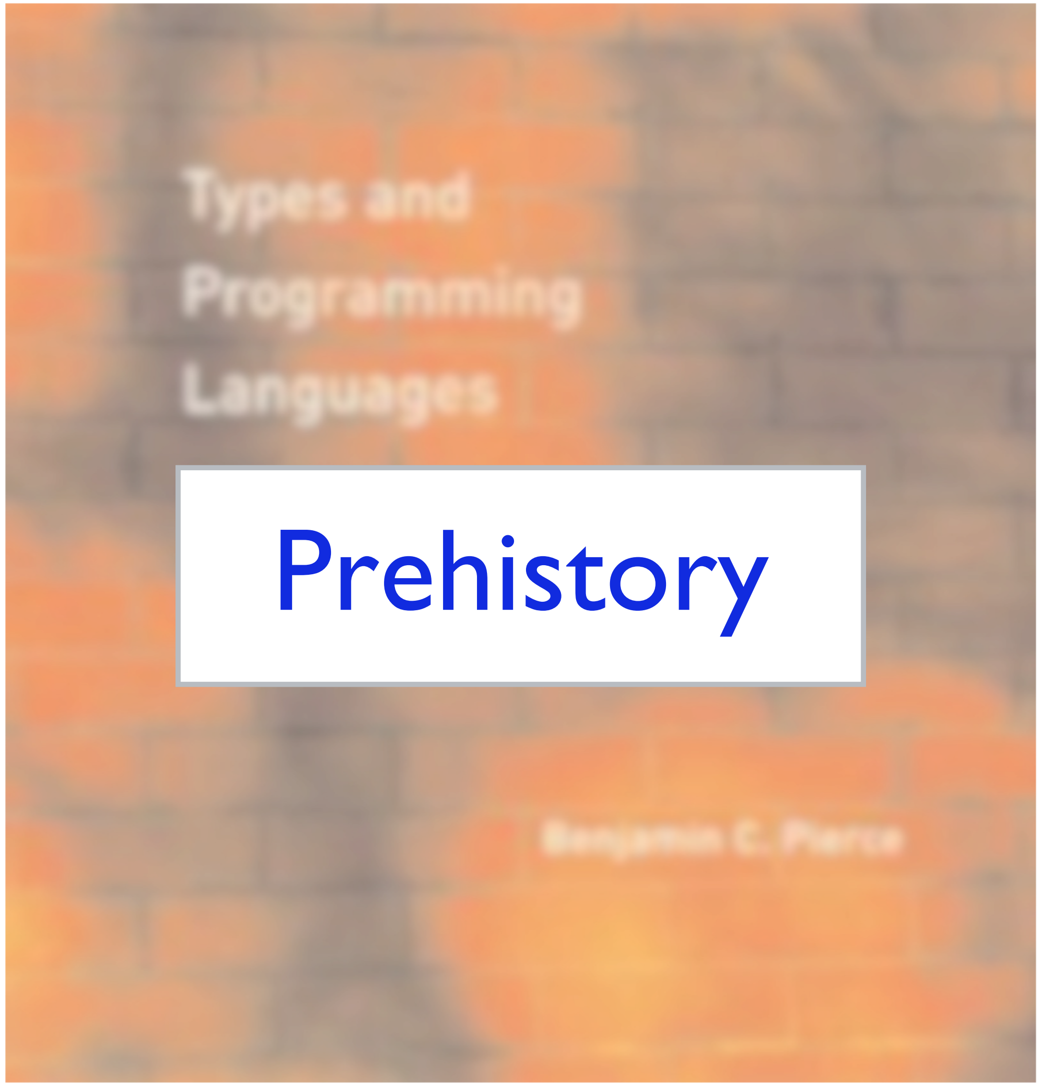
Software Foundations, 15 years on

Benjamin C. Pierce
University of Pennsylvania

Newton Institute workshop on Formal Education
July 2022

Prehistory



The Problem

Large "PL Theory" class

Mixed backgrounds

- In particular, widely varying degrees of mathematical preparation

Great unclarity about what constitutes a proof :-(

➔ Need more TAs!

- … Many more!
- … Maybe even one per student?

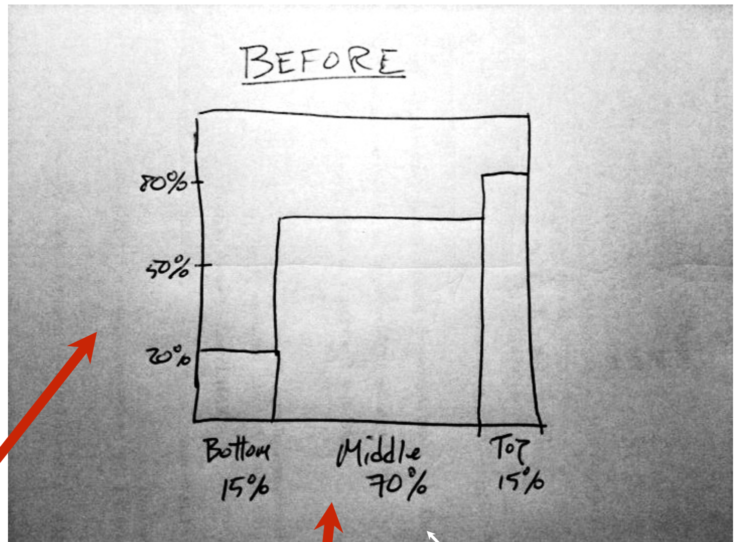
Hmmm…

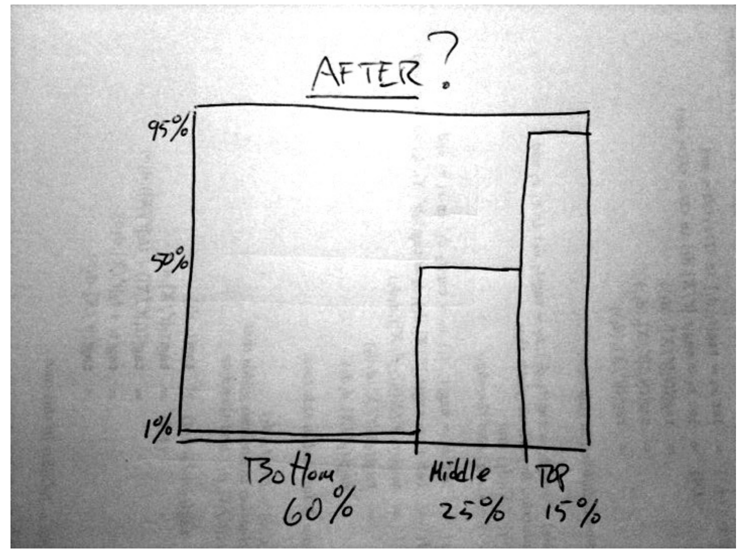
Early history

"Lambda, the Ultimate TA"

- Idea: Use a proof assistant to (sort of) give each student their own TA
- First attempt in Fall 2007
  - Continuous refinement ever since

The Fear



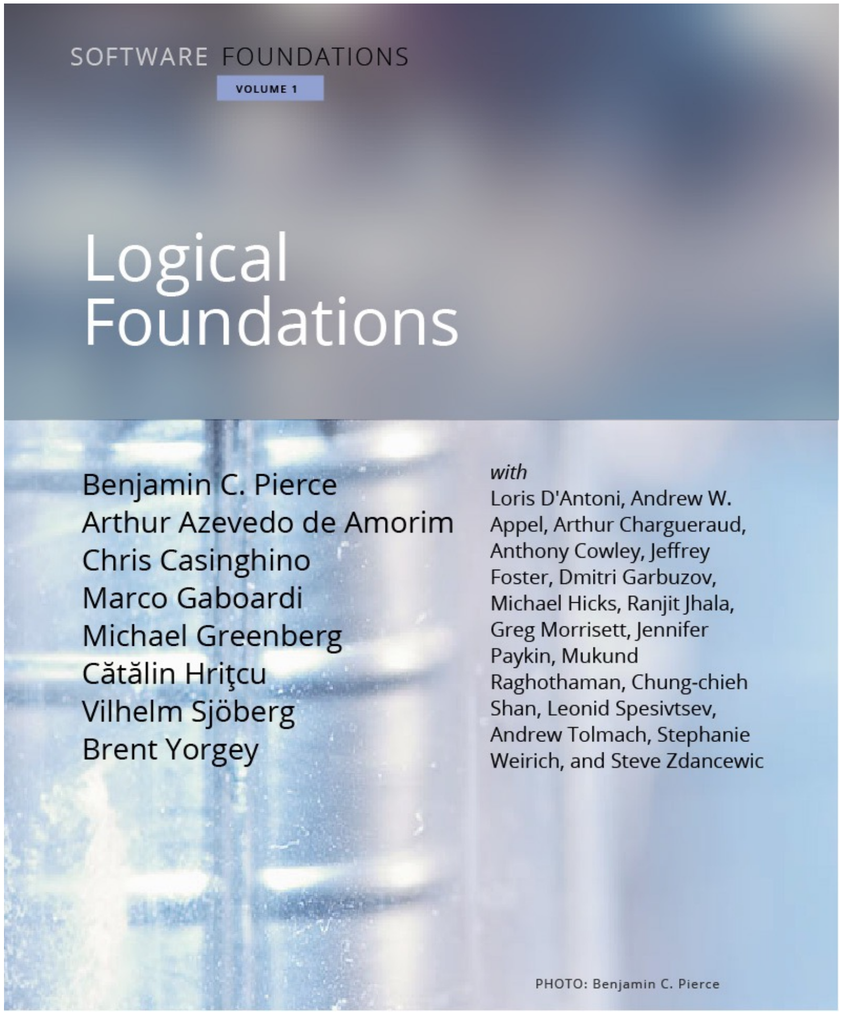


Comprehension

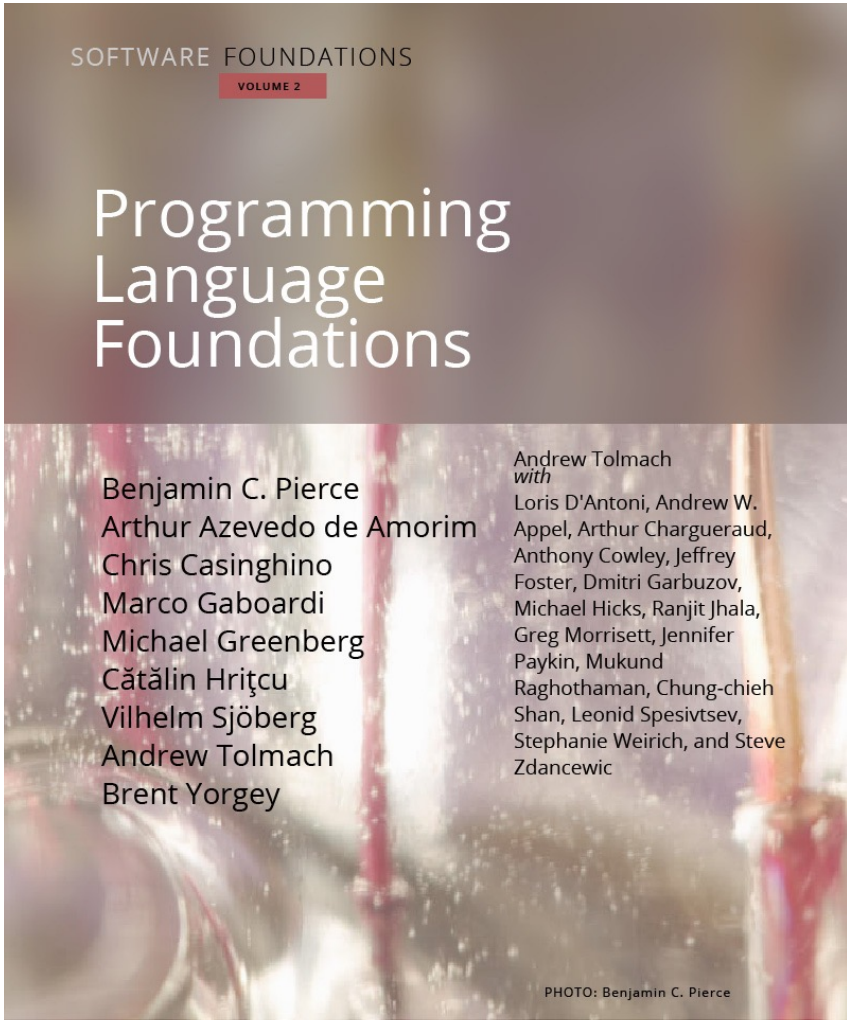
Preparation / aptitude

The Actuality

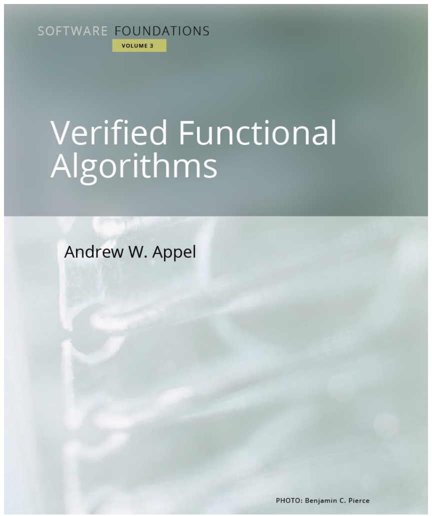
- Bottom 15% does *not* turn into 60%
- Middle 70% learn about as much about PL as before, *and* they get a solid grasp of what induction means
- Top 15% really hone their understanding, both of proofs and of PL theory
- Students actually perform better on paper exams



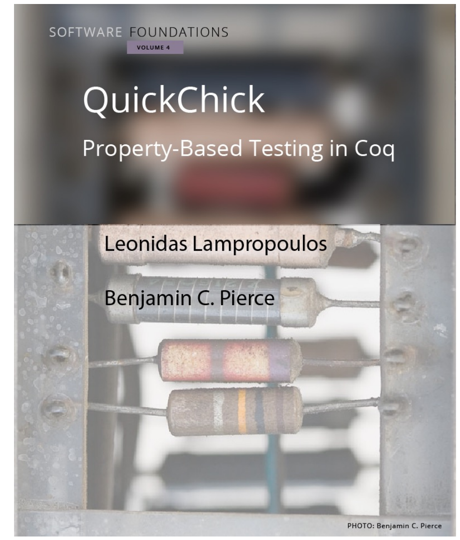
Logical Foundations covers functional programming, basic concepts of logic, computer-assisted theorem proving, and Coq.

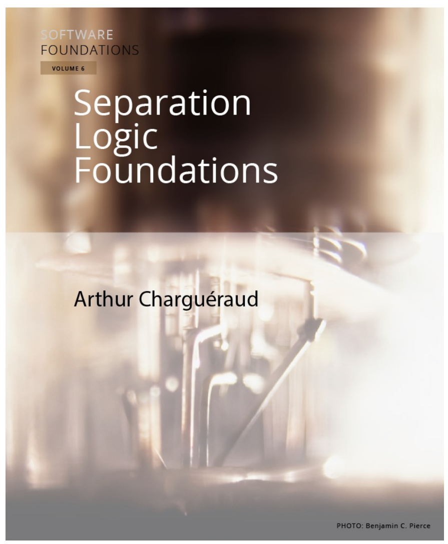


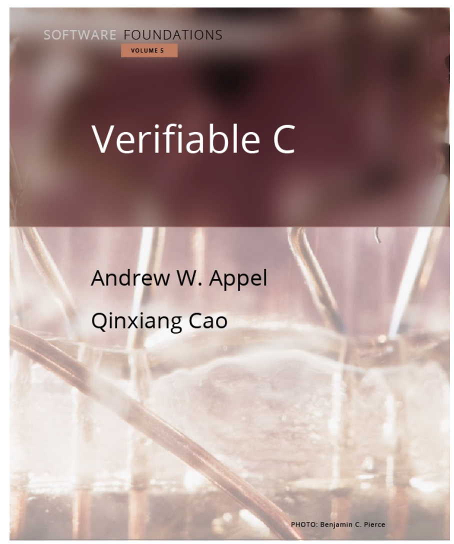
Programming Language Foundations surveys the theory of programming languages, including operational semantics, Hoare logic, and static type systems.

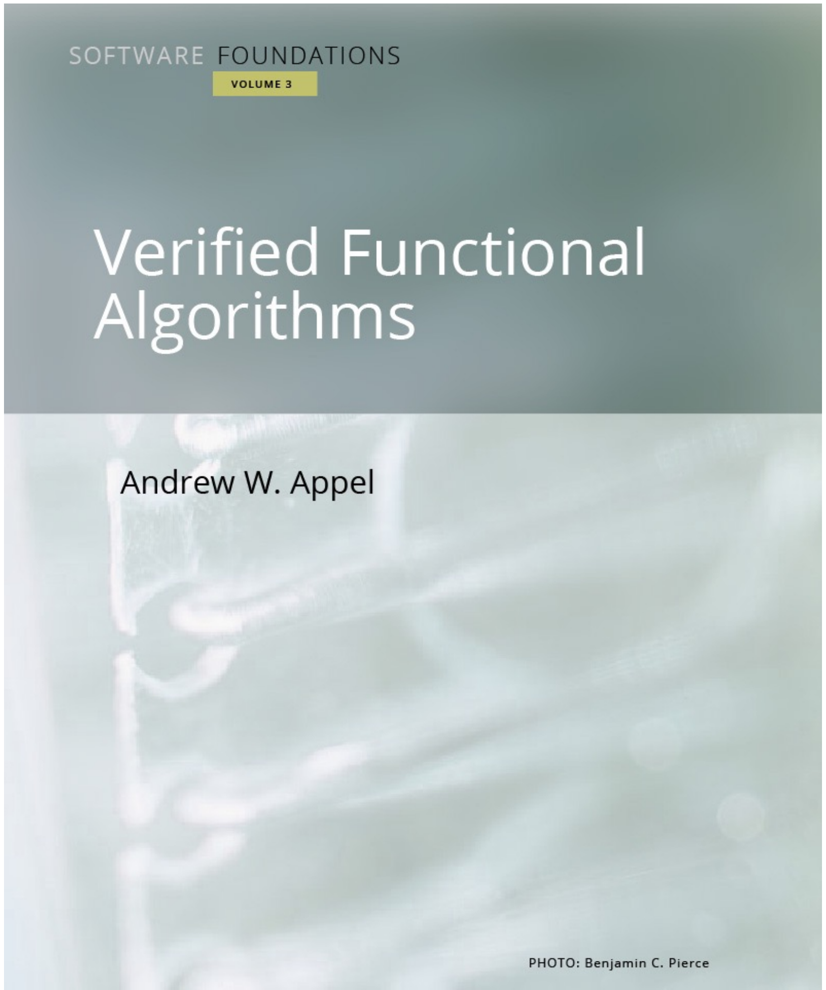


Recent history

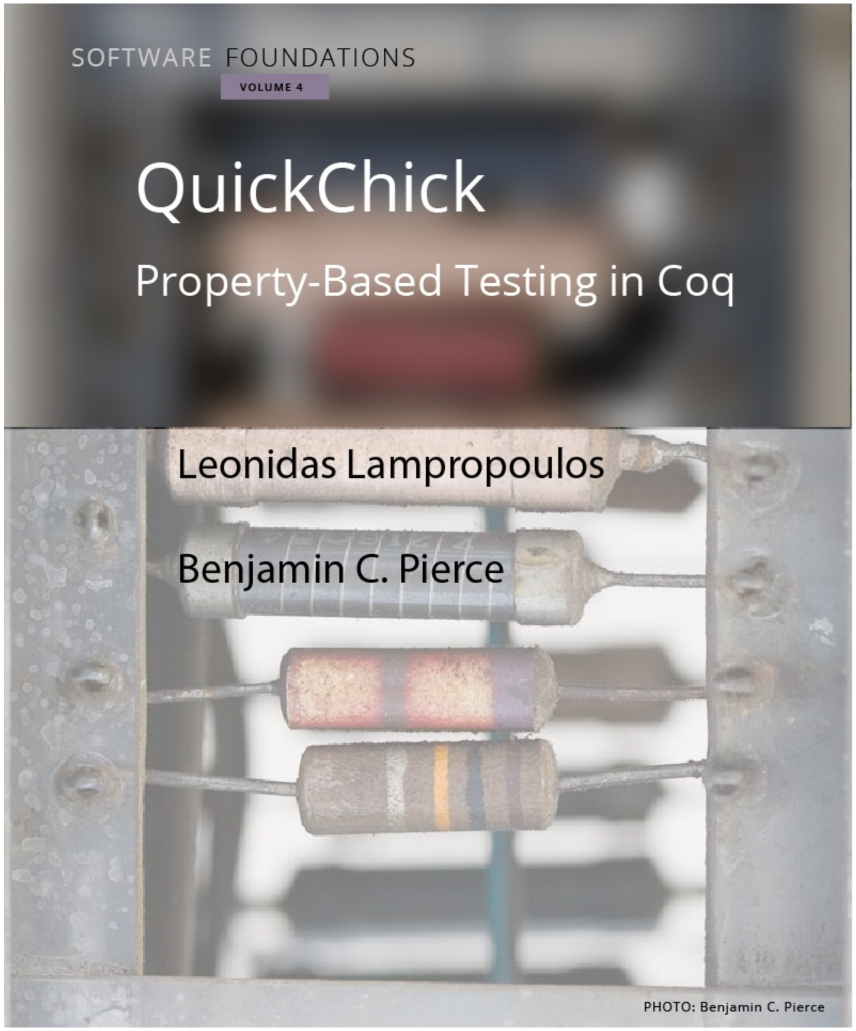






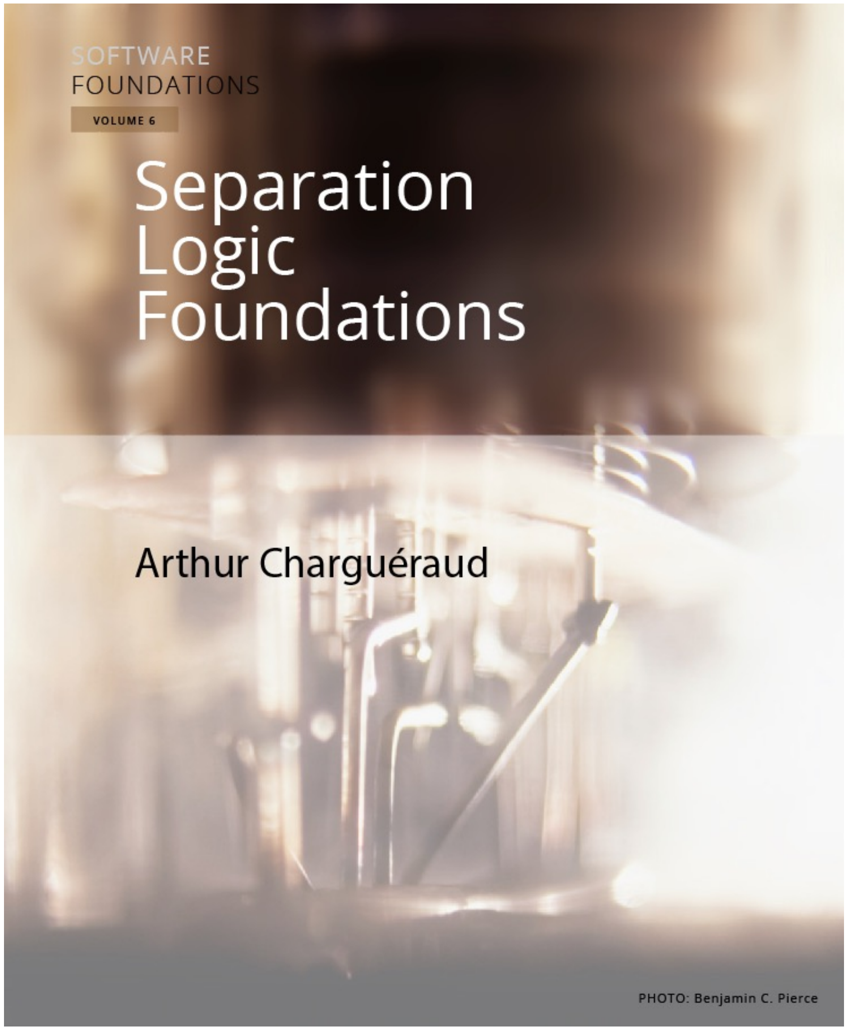


Verified Functional Algorithms shows how a variety of fundamental data structures can be specified and mechanically verified.



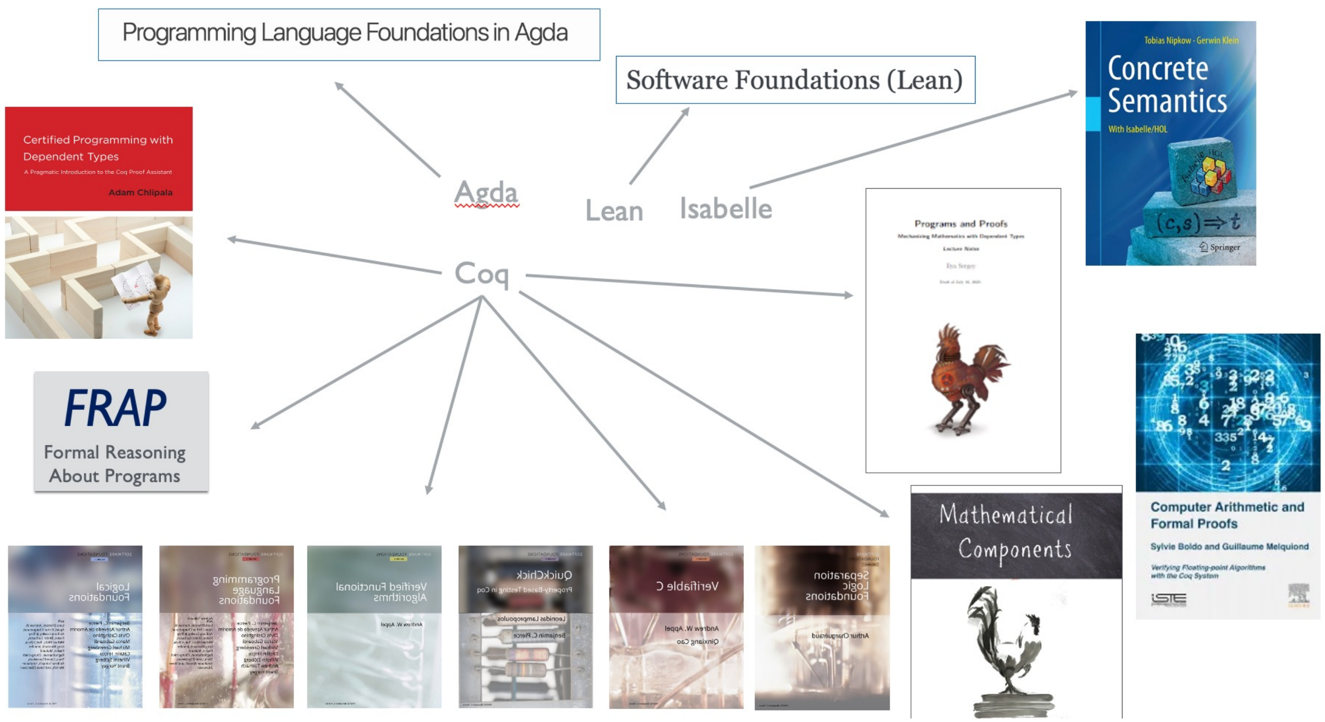
QuickChick: Property-Based Testing in Coq introduces tools for combining randomized property-based testing with formal specification and proof in the Coq ecosystem.

Verifiable C is an extended hands-on tutorial on specifying and verifying real-world C programs using the Princeton Verified Software Toolchain.



Separation Logic Foundations is an in-depth introduction to separation logic—a practical approach to modular verification of imperative programs—and how to build program verification tools on top of it.

Current history



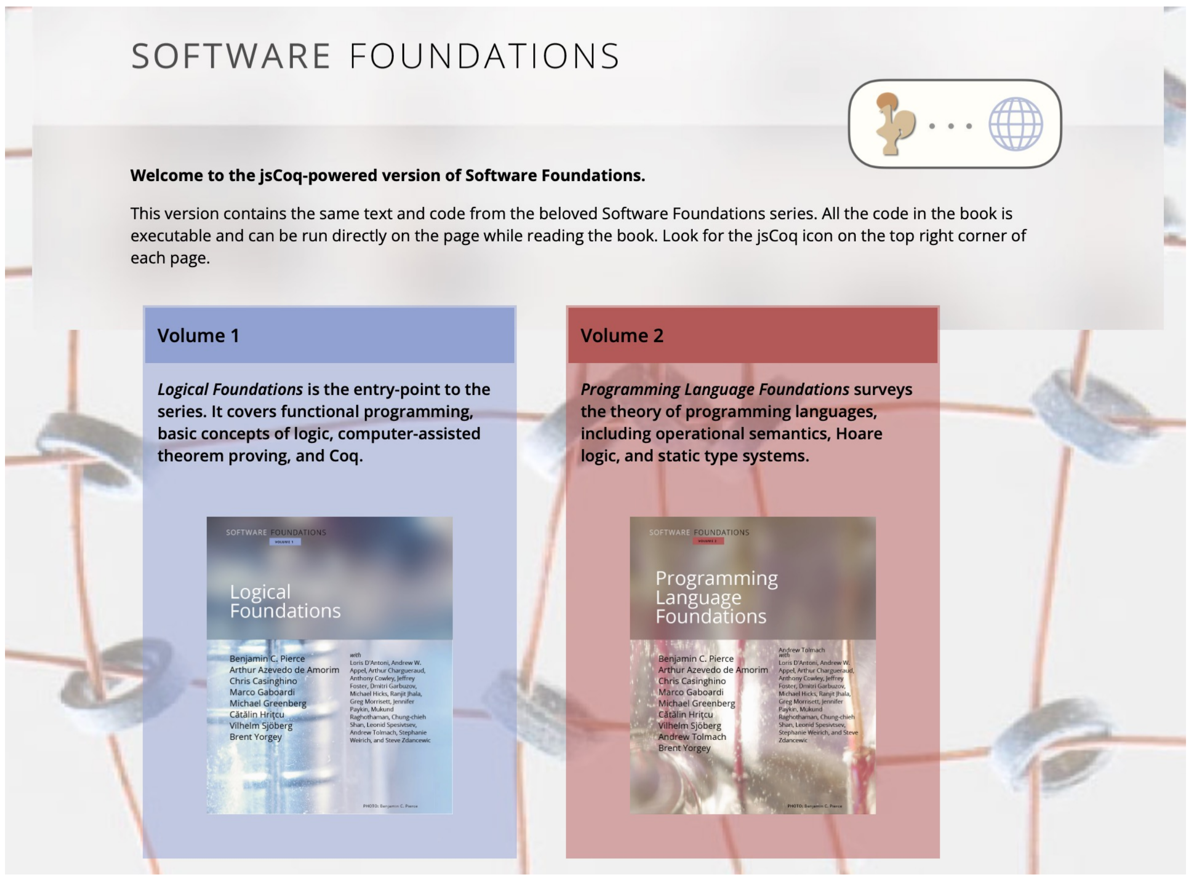
Software Foundations at Penn

- 40-70 students every year
- Mix of undergraduates, MSE students, and PhD students (mostly not studying PL)
- 13 weeks, 23 lectures (80 minutes each), plus 3 review sessions and 3 exams
- Weekly homework assignments (~10-15 hours each)

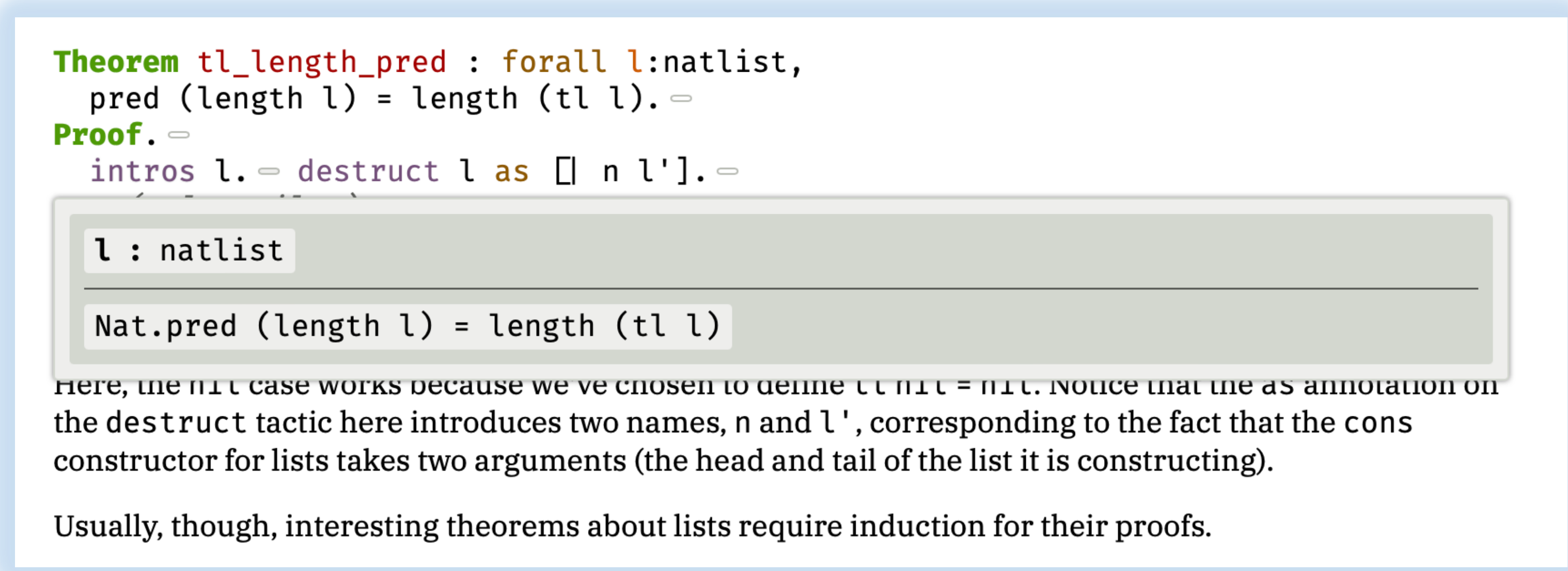
Software Foundations in the Large

- SF is now used at many institutions for undergraduate and graduate teaching
  - Maybe 150-200 students / year?
- 36 contributors to the github repo

Coq in the Browser (Emilio Gallego and Shachar Itzhaky)



Alectryon (Clément Pit-Claudel)

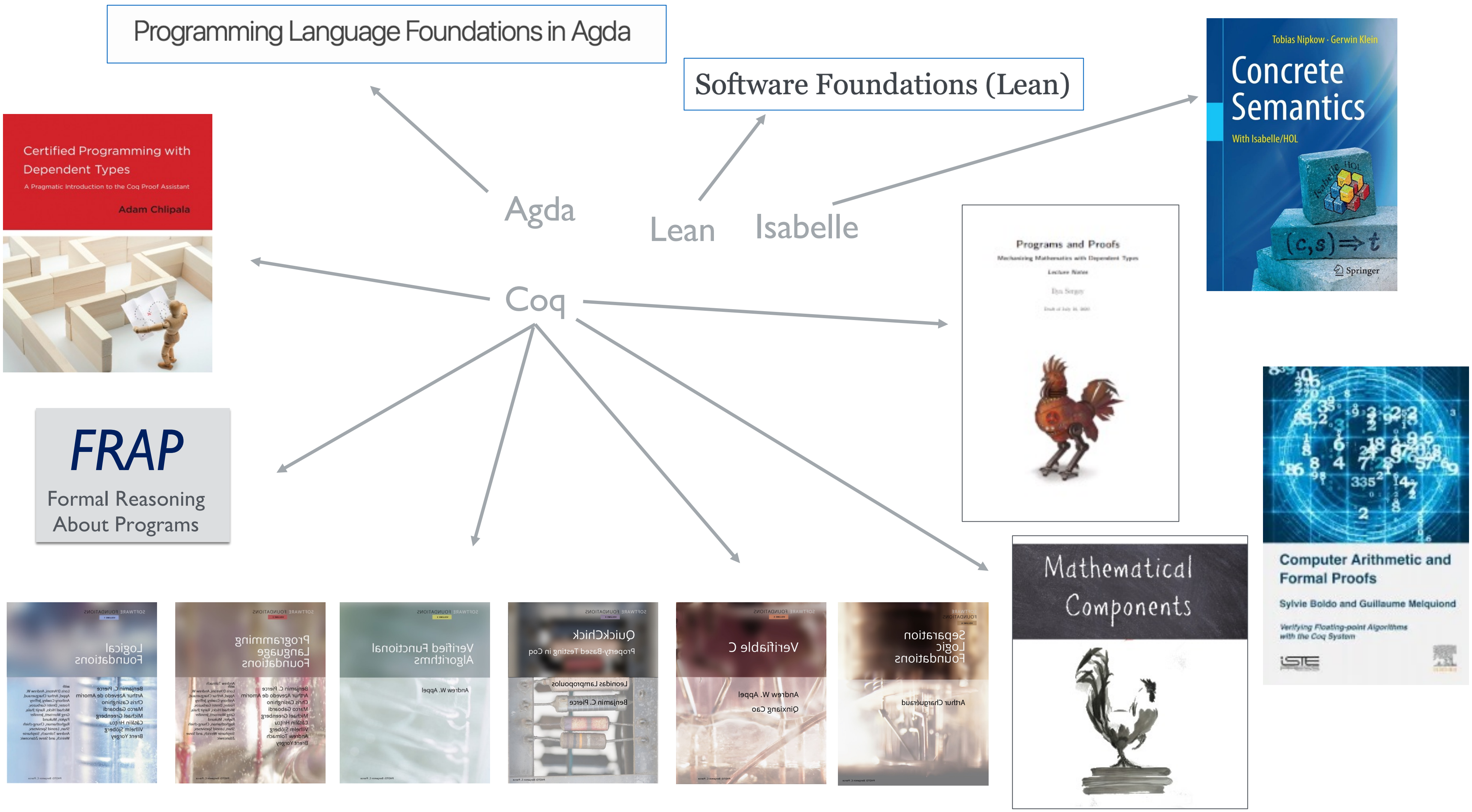


```
Theorem tl_length_pred : forall l:natlist,
  pred (length l) = length (tl l).
Proof.
  intros l. destruct l as [| n l'].
```

```
l : natlist
________________________________
Nat.pred (length l) = length (tl l)
```

Here, the nil case works because we've chosen to define tl nil = nil. Notice that the as annotation on the `destruct` tactic here introduces two names, n and `l'`, corresponding to the fact that the `cons` constructor for lists takes two arguments (the head and tail of the list it is constructing).

Usually, though, interesting theorems about lists require induction for their proofs.



Programming Language Foundations in Agda
Software Foundations (Lean)
Certified Programming with Dependent Types
A Pragmatic Introduction to the Coq Proof Assistant
Adam Chlipala
Agda
Lean
Isabelle
Coq
Tobias Nipkow · Gerwin Klein
Concrete Semantics
With Isabelle/HOL
Springer
Programs and Proofs
Mechanizing Mathematics with Dependent Types
Lecture Notes
FRAP
Formal Reasoning About Programs
Computer Arithmetic and Formal Proofs
Sylvie Boldo and Guillaume Melquiond
Verifying Floating-point Algorithms with the Coq System
ISTE
Mathematical Components

Future history

Future of Software Foundations

- New volumes under construction
  - Discrete math (Greenberg -- see next talk!)
  - Reasoning about interactive programs (Zdancewic)

- Got an idea for another volume? Let's talk…

SF's dual / triple mission

- Teaching logic and PL concepts to a broad audience
- Providing a gentle onramp for learning Coq fundamentals
- Explain how to apply Coq in specific domains

Lessons learned

Teaching with a proof assistant is a Giant Leap!

- Using a proof assistant significantly shapes the way ideas are presented
  - Working "against the grain" of the tool is a really bad idea
- Learning to drive a proof assistant is a significant intellectual challenge

$\Rightarrow$ Restructure entire course around formal proof

Teaching Things to a Proof Assistant is a Giant Pain!

- Developing a machine-checked course is a massive effort
  - Small infelicities cause large headaches
  - No idea if basic definitions are well formulated until the last proof says QED!

- Fortunately, there are several to choose from now :-)

A crucial distinction

(good) informal proofs

Proofs optimized for conveying understanding

vs.

Proofs optimized for conveying certainty

(most) formal proofs

Very challenging to teach
both at the same time!

Flavors of "Formal Proofs"

teach by example

1. Detailed proof in natural language

instructions for writing...

2. Proof-assistant script

program for constructing...

3. Formal proof object

mostly ignore

concentrate here

Is Coq the ultimate TA?

- Almost certainly not the ultimate one!
  - Difficult to identify a "pedagogical subset" of features
  - Easy for students to unwittingly wander into deep waters
  - Lots of subtlety / complexity around even fairly vanilla features
    - E.g., try asking a Coq expert exactly what the "simpl" tactic does…
- …But a pretty good stopgap
  - With careful pedagogy, Coq can absolutely support effective teaching of a broad range of material, even to relatively unsophisticated audiences

Thank you !

Discussion?