CS 3110

Formal Methods

Nate Foster

Spring 2019

Today's music: Theme from *Downton Abbey* by John Lunn

Review

Previously in 3110:

- Functional programming
- Modular programming
- Data structures
- Interpreters

Next unit of course: formal methods

Today:

- Proof assistants
- Functional programming in Coq
- Propositional logic
- Simple proofs about programs

Approaches to validation [lec 11]

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

Less formal: Techniques may miss problems in programs

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?

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

Verification

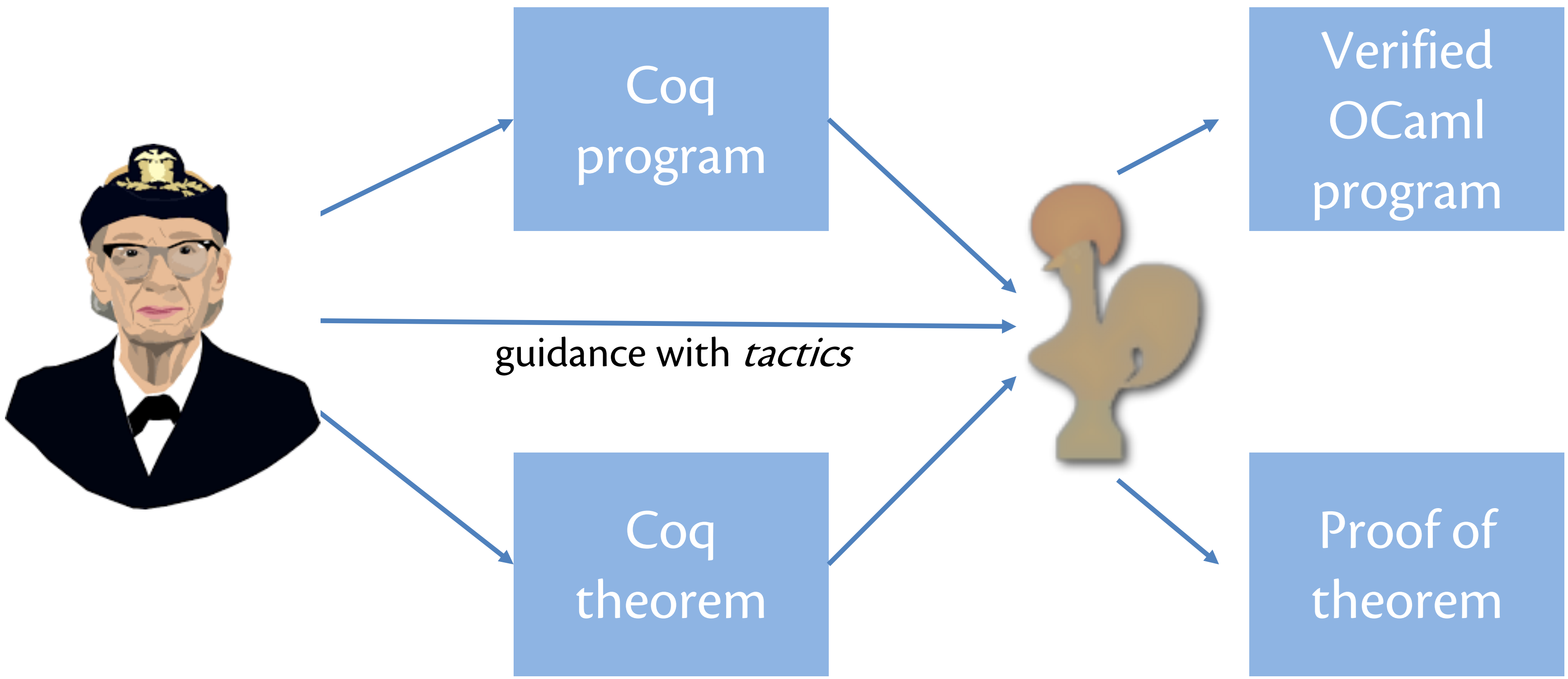
- In the 1970s, scaled to about tens of LOC
- Now, research projects scale to real software:
  - CompCert: verified C compiler
  - seL4: verified microkernel OS
  - Ynot: verified DBMS, web services
  - NetCore: software-defined network controller
- In another 40 years?

Coq

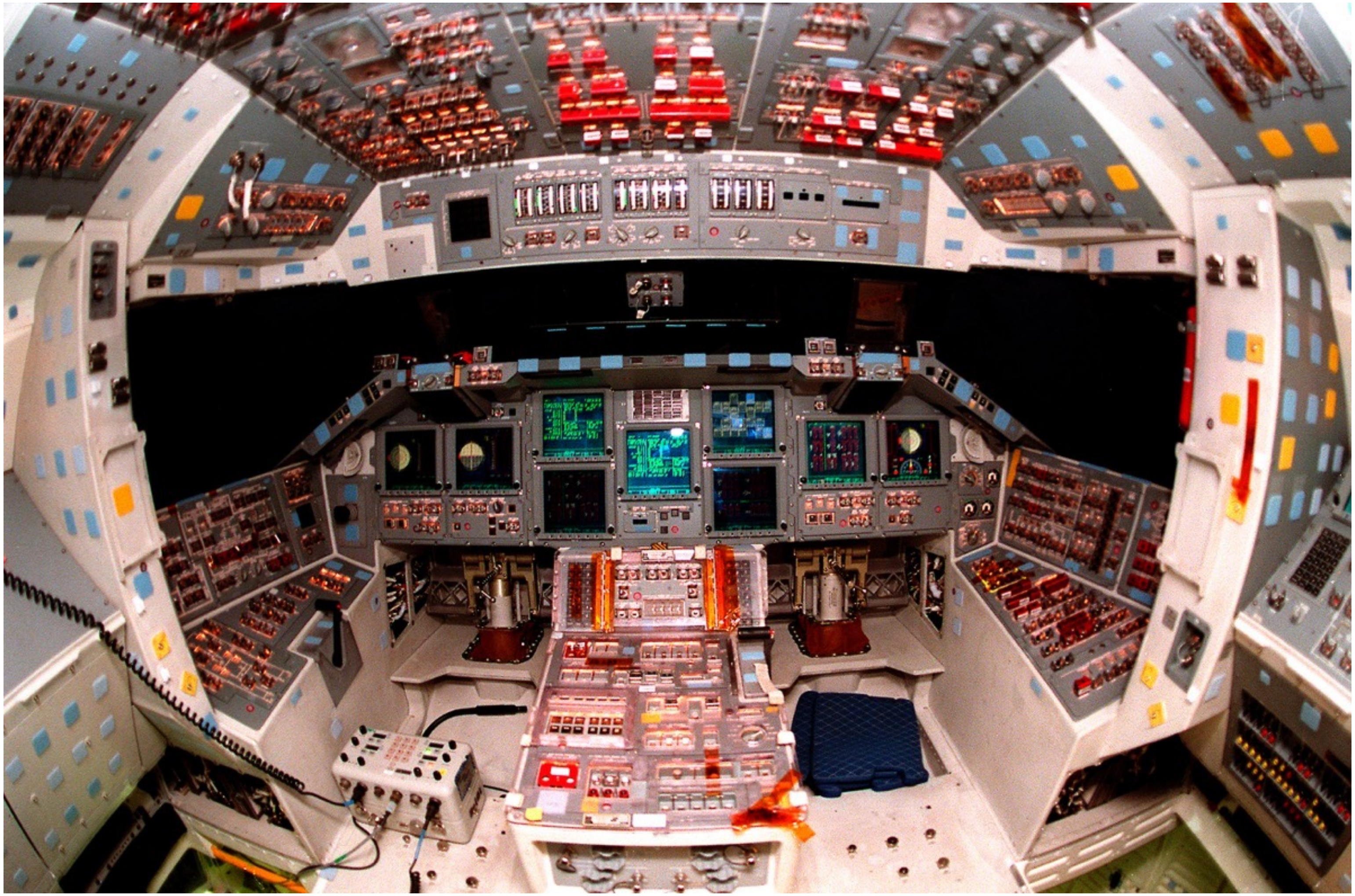
- **1984:** Coquand and Huet implement Coq based on *calculus of inductive constructions*
- **1992:** Coq ported to Caml
- Now implemented in OCaml

Thierry Coquand
1961 –

Coq for program verification



Coq's full system



Subset of Coq we'll use

Our goals

- Write basic functional programs in Coq
  - no side effects, mutability, I/O
- Prove simple theorems in Coq
  - CS 3110 programs: lists, options, trees
  - CS 2800 mathematics: induction, logic

- Non goal: full verification of large programs
- Rather:
  - help you understand what verification involves
  - expose you to the future of functional programming
  - solidify concepts about proof and induction by developing machine-checked proofs

Definitions and Functions

Lists

FUNCTIONAL PROGRAMMING IN COQ

Demo

INDUCTION

Structure of inductive proof

Theorem:
`for all natural numbers n, P(n).`

Proof: `by induction on n`

Case: `n = 0`
Show: `P(0)`

Case: `n = k+1`
IH: `P(k)`
Show: `P(k+1)`

QED

Sum to n

```
let rec sum_to n =
  if n=0 then 0
  else n + sum_to (n-1)
```

$$\sum_{i=0}^{n} i$$

Theorem:

```
for all natural numbers n,
  sum_to n = n * (n+1) / 2.
```

Proof: `by induction on n`

Discussion: What is P? Base case? Inductive case? Inductive hypothesis?

Proof

```
let rec sum_to n =
  if n=0 then 0
  else n + sum_to (n-1)
```

```
P(n) ≡ (sum_to n = n * (n+1) / 2)
```

Case: `n = 0`

Show:

```
P(0)
```

Case: `n = k+1`

IH: `P(k) ≡ sum_to k = k * (k+1) / 2`

Show:

```
P(k+1)
```

QED

INDUCTION ON LISTS

Structure of inductive proof

Theorem:
`for all natural numbers n, P(n).`

Proof: `by induction on n`

Case: `n = 0`
Show: `P(0)`

Case: `n = k+1`
IH: `P(k)`
Show: `P(k+1)`

QED

Structure of inductive proof

Theorem:
`for all lists lst, P(lst).`

Proof: `by induction on lst`

Case: `lst = []`
Show: `P([])`

Case: `lst = h::t`
IH: `P(t)`
Show: `P(h::t)`

QED

Append nil

```
let rec (@) lst1 lst2 =
  match lst1 with
  | []   -> lst2
  | h::t -> h :: (t @ lst2)
```

Theorem:
`for all lists lst, lst @ [] = lst.`

Proof: `by induction on lst`

Discussion: What is P? Base case? Inductive case? Inductive hypothesis?

```
let rec (@) lst1 lst2 =
  match lst1 with
   | []   -> lst2
   | h::t -> h :: (t @ lst2)
```

Base case

`P(lst) ≡ lst @ [] = lst`

Case: `lst = []`

Show:

`P([])`

Case: `lst = h::t`

IH: `P(t) ≡ t @ [] = t`

Show:

`P(h::t)`

QED

INDUCTION ON LISTS IN COQ

Demo

PROPOSITIONAL LOGIC

Logical connectives

- Implication: `p -> p`
- Conjunction: `p /\ p`
- Disjunction: `p \/ p`
- Negation: `~p`

Demo

Implication

output is that proof

```
Print p_implies_p.
p_implies_p =
fun (P : Prop) (P_assumed : P) => P_assumed
     : forall P : Prop, P -> P
```

p_implies_p
is a function

first input is a
proposition

second input is
proof of first input

Coq proofs

are

functional programs

Upcoming events

- [Last night] A8 out!
- [Today] Foster Office Hours @ 1:15pm

This is formal.

THIS IS 3110