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 program
- Coq theorem
- Guidance with tactics
- Verified OCaml program
- Proof of theorem
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
FUNCTIONAL PROGRAMMING IN COQ

Definitions and Functions
Lists
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 $n$

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

**Theorem:**
for all natural numbers $n$,

$$\sum_{i=0}^{n} i = n \times (n+1) / 2.$$

**Proof:** by induction on $n$

**Discussion:** What is $P$? Base case? Inductive case? Inductive hypothesis?
Proof

\( P(n) \equiv (\text{sum\_to } n = n \times (n+1) / 2) \)

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

Case: \( n = k+1 \)
IH: \( P(k) \equiv \text{sum\_to } k = k \times (k+1) / 2 \)
Show:
\( P(k+1) \)

QED

let rec sum_to n =
  if n=0 then 0
  else n + sum_to (n-1)
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 \( \text{lst} \), \( P(\text{lst}) \).

Proof: by induction on \( \text{lst} \)

Case: \( \text{lst} = [] \)
Show: \( P([]) \)

Case: \( \text{lst} = \text{h}::\text{t} \)
IH: \( P(\text{t}) \)
Show: \( P(\text{h}::\text{t}) \)

QED
Append nil

```ocaml
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?
**Base case**

\[ P(lst) \equiv lst @ [] = lst \]

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

Case: \( lst = h::t \)
IH: \( P(t) \equiv t @ [] = t \)
Show:
\[ P(h::t) \]

QED
INDUCTION ON LISTS IN COQ
PROPOSITIONAL LOGIC
Logical connectives

• Implication: $p \implies p$
• Conjunction: $p \land p$
• Disjunction: $p \lor p$
• Negation: $\neg p$
Implication

Print \texttt{p_implies_p}.

\texttt{p_implies_p} = \texttt{fun (P : Prop) (P\_assumed : P) => P\_assumed}

\texttt{forall P : Prop, P -> P}

\texttt{p_implies_p} is a function

first input is a proposition

second input is proof of first input

output is that proof
Coq proofs are functional programs
Upcoming events

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

This is formal.

THIS IS 3110