Interactive Theorem Proving

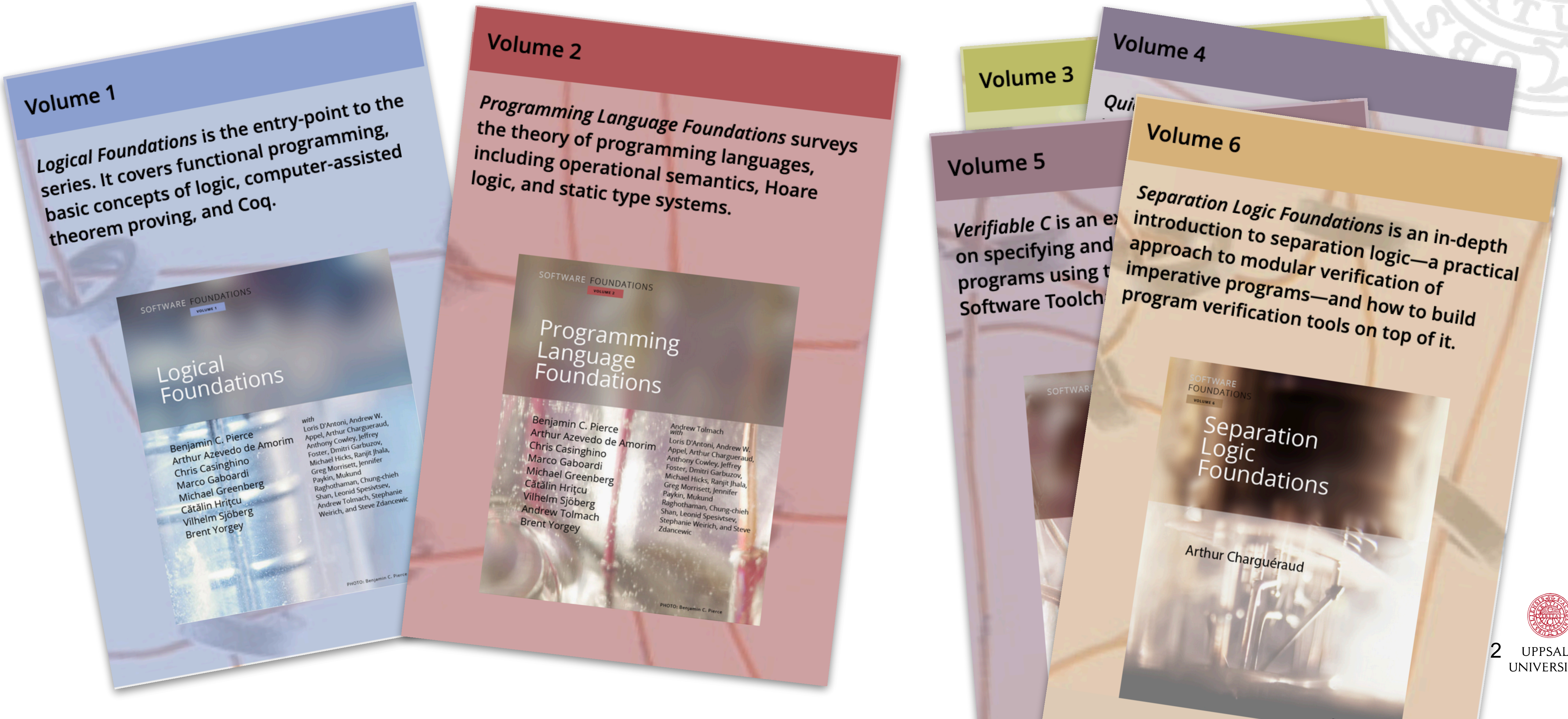
Lecture 4: Where to Now?

Elias Castegren and David Broman

13 June 2024

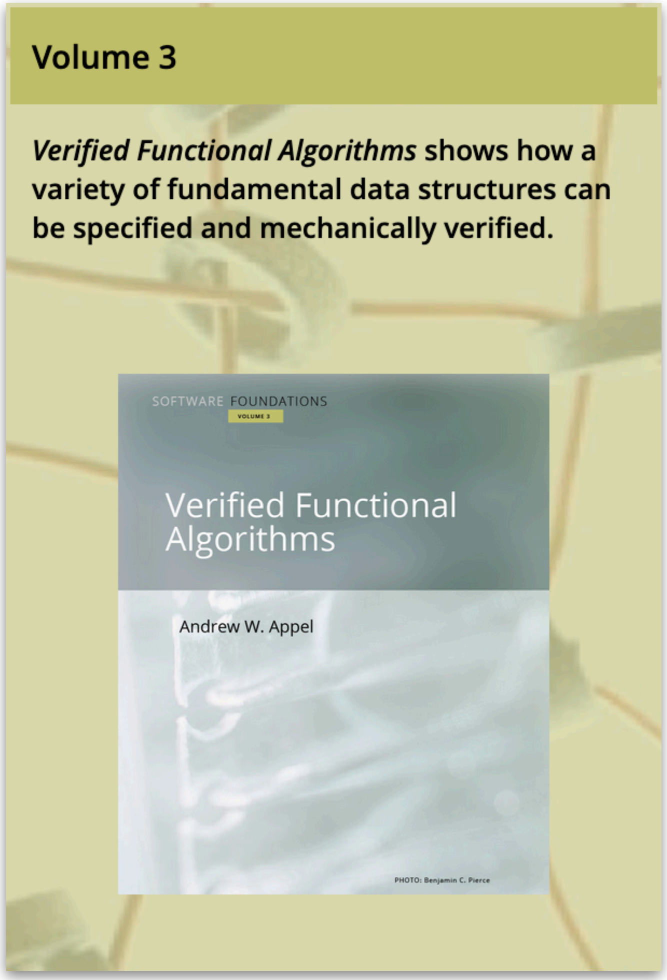
UPPSALA
UNIVERSITET

Software Foundations



UPPSALA UNIVERSITET

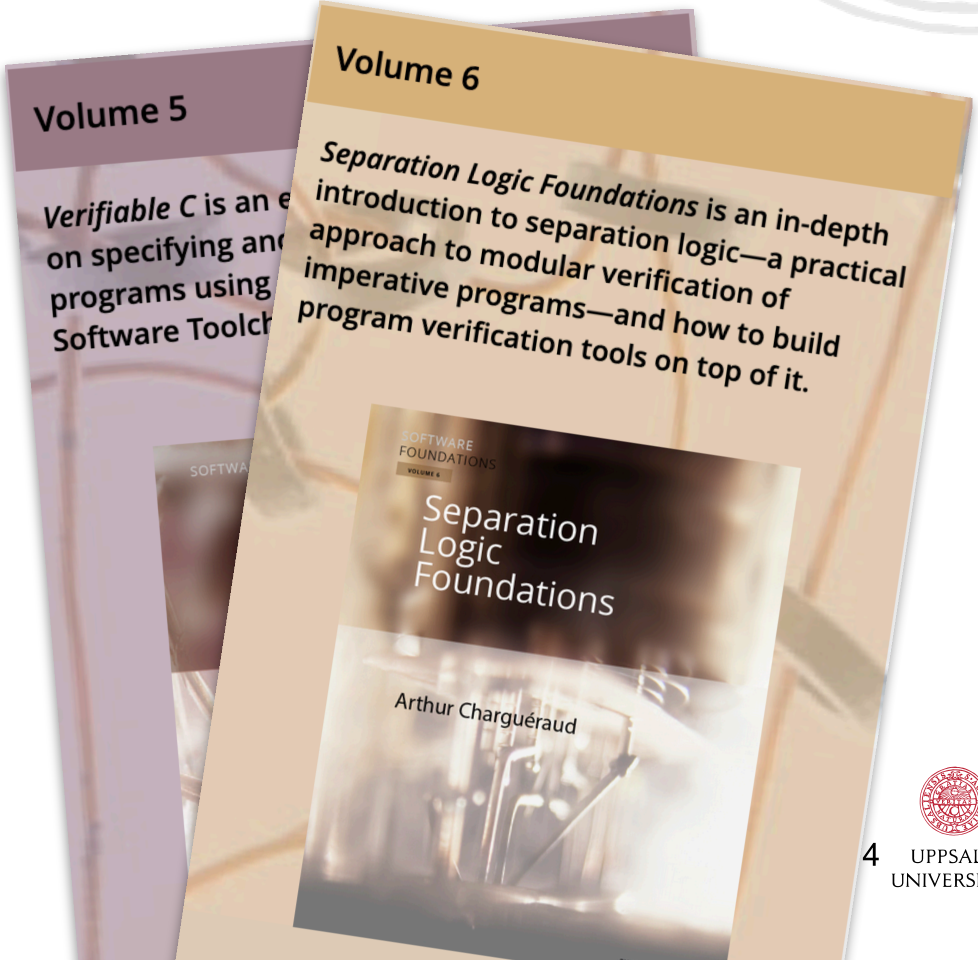
Software Foundations



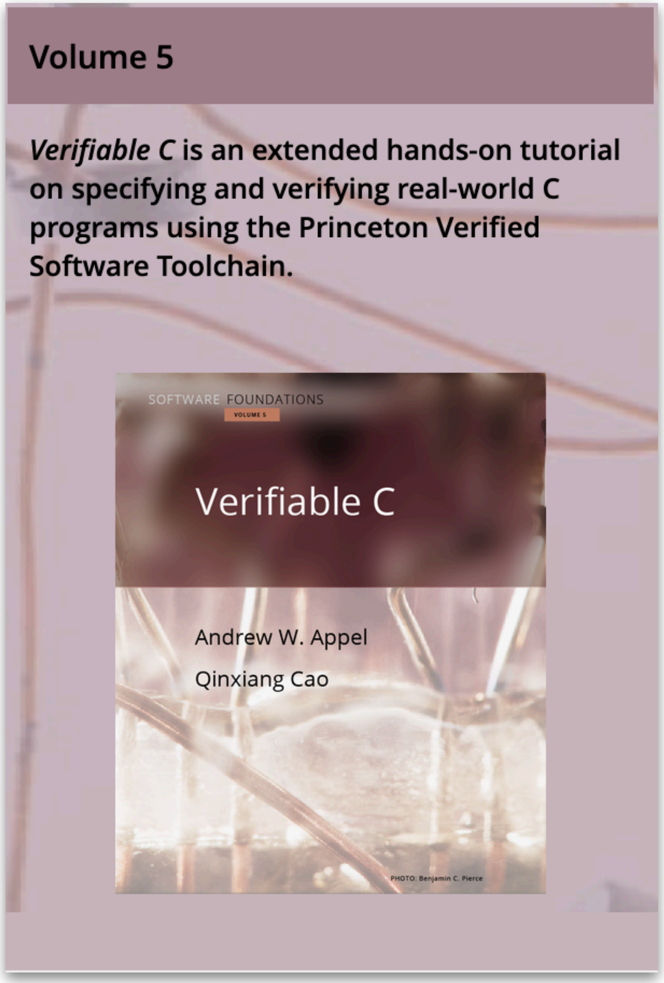
- Verification of functional data structures and algorithms
- Sorting, search trees, balanced trees, priority queues…
- Only depends on Vol. 1
- Lectures available: https://deepspec.org/event/dsss17/lecture_appel.html

Software Foundations

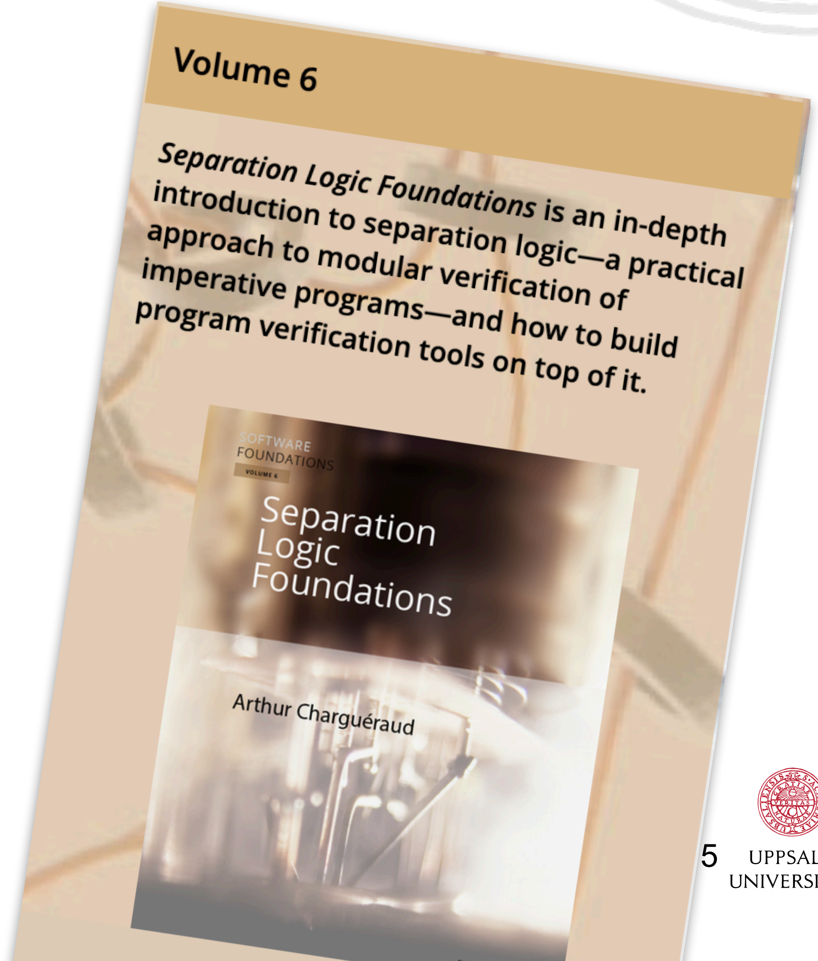
- Combining *property-based testing* with theorem proving
- Introduces *QuickChick*, an implementation of QuickCheck in Coq
- We will look a little bit closer at this today



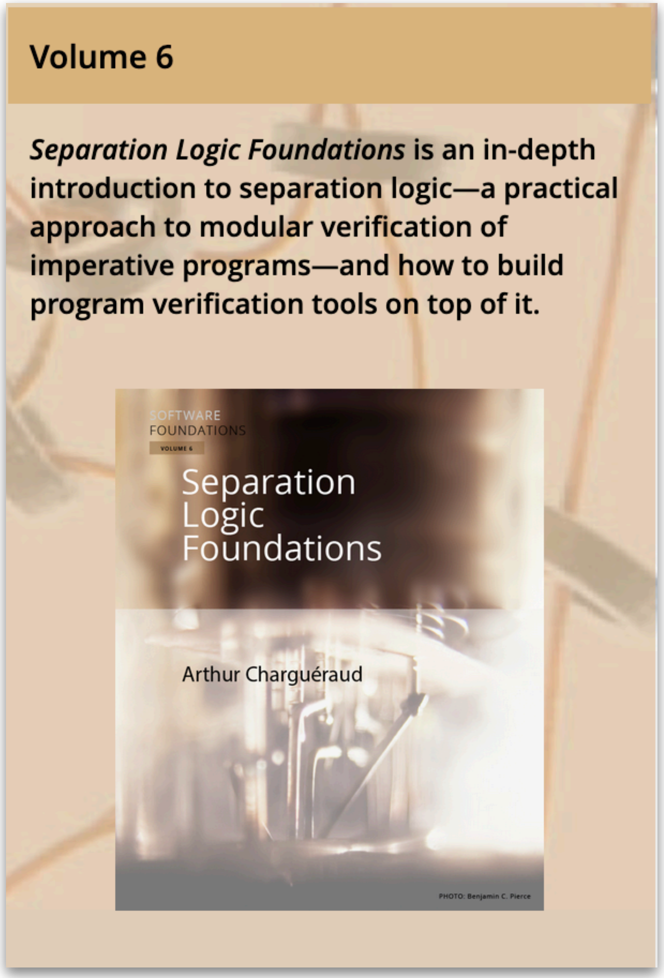
Software Foundations



- Verification of C programs using the Princeton *Verified Software Toolchain*
- Makes use of the verified *CompCert* compiler
- Based on a higher-order impredicative concurrent separation logic and proof system called *Verifiable C*



Software Foundations



- Introduction to separation logic
- Modern extension of the Hoare logic chapters of Vol. 2

- Volume 5 (Verifiable C) focuses on on *using* separation logic
- Volume 6 focuses on *implementing* separation logic

QuickChick: Property-Based Testing in Coq

- Property-based testing provides *randomised testing* of properties

```
Fixpoint remove(x: nat) (l: list nat): list nat :=
    match l with
    | nil => nil
    | y :: ys => if x =? y then ys else y :: remove x ys
    end.

Conjecture remove_property: forall x l, ¬(In x (remove x l)).

QuickChick remove_property.

⟹ 2 [2; 2]
   Failed! After 35 tests and 2 shrinks
```

Overview

- Property-based testing requires four ingredients:
  - An **executable property**
  - **Generators** for generating random input for testing the properties
  - **Printers** for showing data in counterexamples and statistics
  - **Shrinkers** for minimising counterexamples

Simple versions of these (often good enough)
can be derived automatically!

- Generators, printers and shrinkers are based on *type classes*, which helps automation

Type Classes

- Type classes enable ad-hoc polymorphism (overloading)

```
Class Show A : Type :=
  {
    show : A -> string
  }.
```

```
Instance showBool : Show bool :=
  {
    show b :=
      if b then "true" else "false"
  }.
```

```
Instance showNat : Show nat :=
  {
    show := string_of_nat
  }.
```

```
Instance showOption `{Show A} : Show (option A) :=
  {
    show o :=
      match o with
        | None => "None"
        | Some x => "Some " ++ show x
      end
  }.
```

show true $\Longrightarrow$ “true”

show 42 $\Longrightarrow$ “42”

show (Just 10) $\Longrightarrow$ “Just 10”

Type Classes over Properties

- In a language like Coq, we can also define classes of properties

```
Class DecEq A :=
  {
    dec_eq : forall x y : A, {x = y} + {x <> y}
  }.
```

```
Instance NatDecEq : DecEq nat.
  constructor.
  decide equality.
Defined.
```

```
Class Decidable (P : Prop) :=
  {
    dec : {P} + {~P}
  }.
```

```
if dec_eq 2 3 then true else false ⟹ false

if @dec (2 = 3) _ then true else false ⟹ false
```

```
Instance DecEqDec `{eqDec: DecEq A} {x y : A}: Decidable (x = y).
  constructor.
  destruct eqDec as [eqDec].
  destruct (eqDec x y); auto.
Defined.
```

Type Classes in QuickChick

- Property-based testing requires four ingredients:
  - ✓ An **executable property** Decidability makes even propositions executable
  - — **Generators** for generating random input for testing the properties
  - ✓ **Printers** for showing data in counterexamples and statistics
  - — **Shrinkers** for minimising counterexamples

Type Classes in QuickChick

- Property-based testing requires two additional ingredients:
  - **Generators** for generating random input for testing the properties

```
Inductive G A :=
| MkGen: (nat -> RandomSeed -> A) -> G A.
```

```
Class Gen : (A : Type) :=
  {
    arbitrary : G A
  }.
```

```
Instance GMonad : `{Monad G} :=
  {
    ret := returnGen: A -> G A
    bind := bindGen: G A -> (A -> G B) -> G B
  }.
```

  - **Shrinkers** for minimising counterexamples

```
Class Shrink : (A : Type) :=
  {
    shrink : A -> List A
  }.
```

Example: Binary Trees

- A generator and shrinker for binary trees

```
Fixpoint genTreeSized {A} (sz: nat) (g: G A) : G (Tree A) :=
  match sz with
    | O => ret Leaf
    | S sz' =>
        oneOf [
          ret Leaf ;
          liftM3 Node g (genTreeSized sz' g) (genTreeSized sz' g)
        ]
  end.
```

oneOf: list (G A) -> G A

```
Fixpoint shrinkTree {A} (s: A -> list A) (t: Tree A): list (Tree A) :=
  match t with
    | Leaf => []
    | Node x l r => [l] ++ [r] ++
                    map (fun x' => Node x' l r) (s x) ++
                    map (fun l' => Node x l' r) (shrinkTree s l) ++
                    map (fun r' => Node x l r') (shrinkTree s r)
  end.
```

Generating Useful Data

- Randomly generated data will contain repetitions
  - QuickChick allows you to skew the distribution in generators

```
oneOf: list (G A) -> G A
 freq: list (nat * G A) -> G A
```

- Often we are only interested in data with a particular shape
  - QuickChick lets you discard data based on preconditions

```
Definition bst_insert_spec t x := isBST t ==> isBST (insert x t)
```

  - Often it is more efficient to write better generators!

Main takeaways

- Testing helps proving! You can quickly check your properties before proving them correct

- Proving helps testing! You are already specifying properties of your data which can be used to generate better input data

```
Definition preservation := forall e1 e2 t,
    has_type e1 t -> steps e1 e2 -> has_type e2 t
```

There is a definition for this that can be used to generate well-typed expressions e1

There is a definition for this that can be used to generate expressions e2 given an input e1

- QuickChick is an active research project

Questions so far?

UPPSALA
UNIVERSITET

OOlong: an Extensible Concurrent Object Calculus

Published 2018

My first “real” Coq project

UPPSALA UNIVERSITET

Modelling Reality

http://www.nicehomezone.com/wp-content/uploads/ [illegible]

Modelling Programming Languages

Featherweight Java

ClassicJava

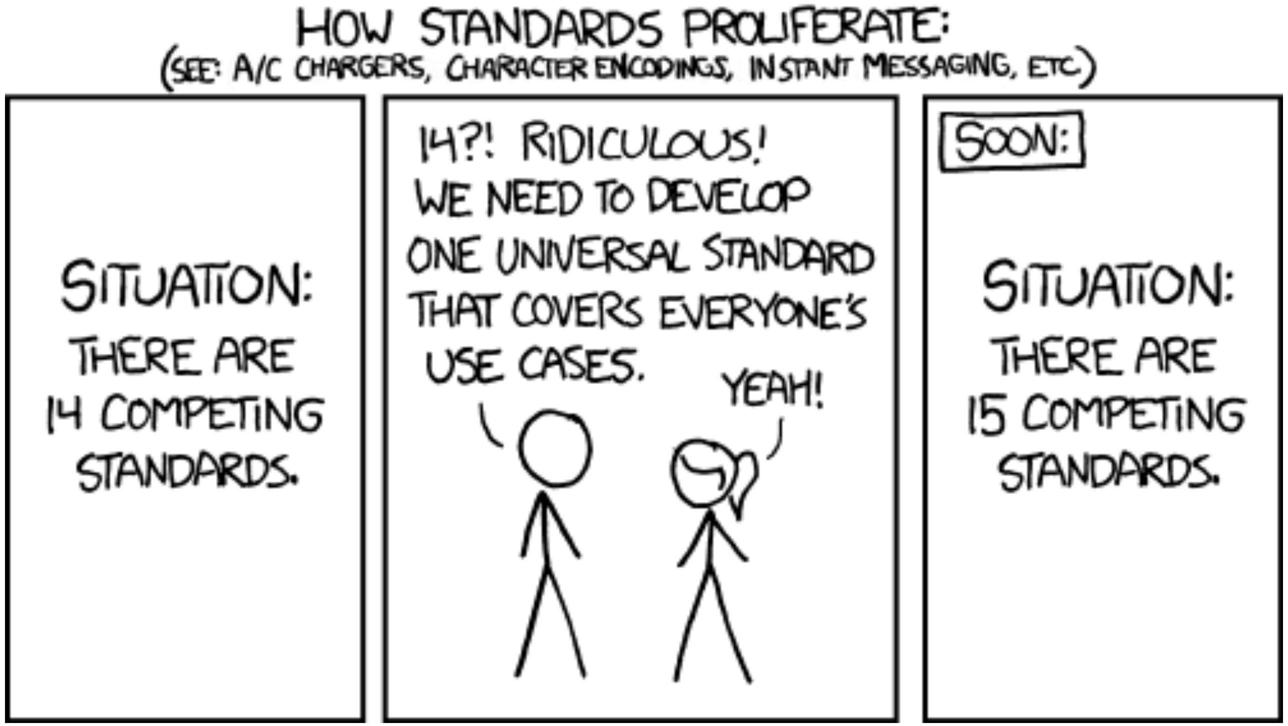
ConcurrentJava

Lightweight Java

Middleweight Java

Welterweight Java

Java™



https://imgs.xkcd.com/comics/standards.png

Different Calculi have Different Level of Detail

	FJ	ClJ	ConJ	MJ	LJ	WJ
State		×	×	×	×	×
Statements				×	×	×
Expressions	×	×	×	×		
Class Inheritance	×	×	×	×	×	×
Interfaces		×				
Concurrency			×			×
Stack				×		×
Mechanised	×*				×	
LaTeX sources					×	×

OOlong — Example Program

```
interface Counter {
  add(x : int) : unit
  get() : int
}

class Cell implements Counter {
  cnt : int
  def add(n : int) : unit {
    this.cnt = this.cnt + n
  }
  def get() : int {
    this.cnt
  }
}
```

```
let cell = new Cell in
  finish {
    async {
      lock(cell) in cell.add(1)
    }
    async {
      lock(cell) in cell.add(2)
    }
  };
  cell.get() // Read 3
```

OOlong — Syntax

P	$::=$	$Ids\ Cds\ e$	(Programs)
Id	$::=$	**interface** I $\{Msigs\}$	(Interfaces)
	$\mid$	**interface** I **extends** I_1, I_2	
Cd	$::=$	**class** C **implements** I $\{Fds\ Mds\}$	(Classes)
$Msig$	$::=$	$m(x : t_1) : t_2$	(Signatures)
Fd	$::=$	$f : t$	(Fields)
Md	$::=$	**def** $Msig$ $\{e\}$	(Methods)
e	$::=$	$v \mid x \mid x.f \mid x.f = e$	(Expressions)
	$\mid$	$x.m(e) \mid$ **let** $x = e_1$ **in** $e_2 \mid$ **new** $C \mid (t)\ e$	
	$\mid$	**finish**$\{$**async**$\{e_1\}$ **async**$\{e_2\}\}$; e_3	
	$\mid$	**lock**(x) **in** $e \mid$ **locked**$_\iota\{e\}$	
v	$::=$	**null** $\mid \iota$	(Values)
t	$::=$	$C \mid I \mid$ **Unit**	(Types)
Γ	$::=$	$\epsilon \mid \Gamma, x : t \mid \Gamma, \iota : C$	(Typing environment)

Run-time constructs

UPPSALA UNIVERSITET

OOlong — Static Semantics

$\boxed{\Gamma \vdash e : t}$ *(Typing Expressions)*

WF-VAR
$$\frac{\vdash \Gamma \qquad \Gamma(x) = t}{\Gamma \vdash x : t}$$

WF-LET
$$\frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma, x : t_1 \vdash e_2 : t}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : t}$$

WF-CALL
$$\frac{\Gamma(x) = t_1 \qquad \Gamma \vdash e : t_2 \qquad \textbf{msigs}\,(t_1)(m) = y : t_2 \to t}{\Gamma \vdash x.m(e) : t}$$

WF-CAST
$$\frac{\Gamma \vdash e : t' \qquad t' <: t}{\Gamma \vdash (t)e : t}$$

WF-SELECT
$$\frac{\Gamma \vdash x : C \qquad \textbf{fields}\,(C)(f) = t}{\Gamma \vdash x.f : t}$$

WF-UPDATE
$$\frac{\Gamma \vdash x : C \qquad \Gamma \vdash e : t \qquad \textbf{fields}\,(C)(f) = t}{\Gamma \vdash x.f = e : \textbf{Unit}}$$

WF-NEW
$$\frac{\vdash \Gamma \qquad \vdash C}{\Gamma \vdash \textbf{new}\ C : C}$$

WF-LOC
$$\frac{\vdash \Gamma \qquad \Gamma(\iota) = C \qquad C <: t}{\Gamma \vdash \iota : t}$$

WF-NULL
$$\frac{\vdash \Gamma \qquad \vdash t}{\Gamma \vdash \textbf{null} : t}$$

WF-FJ
$$\frac{\Gamma = \Gamma_1 + \Gamma_2 \qquad \Gamma_1 \vdash e_1 : t_1 \qquad \Gamma_2 \vdash e_2 : t_2 \qquad \Gamma \vdash e : t}{\Gamma \vdash \textbf{finish}\ \{\ \textbf{async}\ \{\ e_1\ \}\ \textbf{async}\ \{\ e_2\ \}\ \}; e : t}$$

WF-LOCK
$$\frac{\Gamma \vdash x : t_2 \qquad \Gamma \vdash e : t}{\Gamma \vdash \textbf{lock}(x)\ \textbf{in}\ e : t}$$

WF-LOCKED
$$\frac{\Gamma \vdash e : t \qquad \Gamma(\iota) = t_2}{\Gamma \vdash \textbf{locked}_\iota\{e\} : t}$$

$\boxed{\vdash P : t \quad \vdash Id \quad \vdash Cd \quad \vdash Fd \quad \vdash Md}$ *(Well-formed program)*

WF-PROGRAM
$$\frac{\forall\, Id \in Ids.\ \vdash Id \qquad \forall\, Cd \in Cds.\ \vdash Cd \qquad \epsilon \vdash e : t}{\vdash Ids\ Cds\ e : t}$$

WF-INTERFACE
$$\frac{\forall\, m(x : t) : t' \in Msigs.\ \vdash t \wedge \vdash t'}{\vdash \textbf{interface}\ I\ \{\ Msigs\ \}}$$

WF-INTERFACE-EXTENDS
$$\frac{\vdash I_1 \qquad \vdash I_2}{\vdash \textbf{interface}\ I\ \textbf{extends}\ I_1, I_2}$$

WF-CLASS
$$\frac{\forall\, m(x : t) : t' \in \textbf{msigs}\,(I).\textbf{def}\ m(x : t) : t'\ \{\ e\ \} \in Mds \qquad \forall\, Fd \in Fds.\ \vdash Fd \qquad \forall\, Md \in Mds.\textbf{this} : C \vdash Md}{\vdash \textbf{class}\ C\ \textbf{implements}\ I\ \{\ Fds\ Mds\ \}}$$

WF-FIELD
$$\frac{\vdash t}{\vdash f : t}$$

WF-METHOD
$$\frac{\textbf{this} : C, x : t \vdash e : t'}{\textbf{this} : C \vdash \textbf{def}\ m(x : t) : t'\ \{\ e\ \}}$$

OOlong — Runtime Configuration

cfg	$::=$	$\langle H; V; T \rangle$	*(Configuration)*
H	$::=$	$\epsilon \mid H, \iota \mapsto obj$	*(Heap)*
V	$::=$	$\epsilon \mid V, x \mapsto v$	*(Variable map)*
T	$::=$	$(\mathcal{L}, e) \mid T_1 \mid\mid T_2 \rhd e \mid \textbf{EXN}$	*(Threads)*
obj	$::=$	(C, F, L)	*(Objects)*
F	$::=$	$\epsilon \mid F, f \mapsto v$	*(Field map)*
L	$::=$	**locked** | **unlocked**	*(Lock status)*
EXN	$::=$	**NullPointerException**	*(Exceptions)*

OOlong — Dynamic Semantics

$\boxed{cfg_1 \hookrightarrow cfg_2}$ *(Evaluation of expressions)*

DYN-EVAL-CONTEXT
$$\frac{\langle H; V; (\mathcal{L}, e)\rangle \hookrightarrow \langle H'; V'; (\mathcal{L}', e')\rangle}{\langle H; V; (\mathcal{L}, E[e])\rangle \hookrightarrow \langle H'; V'; (\mathcal{L}', E[e'])\rangle}$$

DYN-EVAL-VAR
$$\frac{V(x) = v}{\langle H; V; (\mathcal{L}, x)\rangle \hookrightarrow \langle H; V; (\mathcal{L}, v)\rangle}$$

DYN-EVAL-LET
$$\frac{x' \textbf{ fresh} \qquad V' = V[x' \mapsto v] \qquad e' = e[x \mapsto x']}{\langle H; V; (\mathcal{L}, \textbf{let } x = v \textbf{ in } e)\rangle \hookrightarrow \langle H; V'; (\mathcal{L}, e')\rangle}$$

DYN-EVAL-CALL
$$\frac{\begin{array}{c} V(x) = \iota \qquad H(\iota) = (C, F, L) \\ \textbf{methods}(C)(m) = y : t_2 \to t, e \\ \textbf{this}' \textbf{ fresh} \qquad y' \textbf{ fresh} \\ V' = V[\textbf{this}' \mapsto \iota][y' \mapsto v] \\ e' = e[\textbf{this} \mapsto \textbf{this}'][y \mapsto y'] \end{array}}{\langle H; V; (\mathcal{L}, x.m(v))\rangle \hookrightarrow \langle H; V'; (\mathcal{L}, e')\rangle}$$

DYN-EVAL-CAST
$$\frac{}{\langle H; V; (\mathcal{L}, (t)v)\rangle \hookrightarrow \langle H; V; (\mathcal{L}, v)\rangle}$$

DYN-EVAL-SELECT
$$\frac{\begin{array}{c} V(x) = \iota \qquad H(\iota) = (C, F, L) \\ F(f) = v \end{array}}{\langle H; V; (\mathcal{L}, x.f)\rangle \hookrightarrow \langle H; V; (\mathcal{L}, v)\rangle}$$

DYN-EVAL-UPDATE
$$\frac{\begin{array}{c} V(x) = \iota \qquad H(\iota) = (C, F, L) \\ H' = H[\iota \mapsto (C, F[f \mapsto v], L)] \end{array}}{\langle H; V; (\mathcal{L}, x.f = v)\rangle \hookrightarrow \langle H'; V; (\mathcal{L}, \textbf{null})\rangle}$$

DYN-EVAL-NEW
$$\frac{\begin{array}{c} \textbf{fields}(C) \equiv f_1 : t_1, .., f_n : t_n \\ F \equiv f_1 \mapsto \textbf{null}, .., f_n \mapsto \textbf{null} \\ \iota \textbf{ fresh} \qquad H' = H[\iota \mapsto (C, F, \textbf{unlocked})] \end{array}}{\langle H; V; (\mathcal{L}, \textbf{new } C)\rangle \hookrightarrow \langle H'; V; (\mathcal{L}, \iota)\rangle}$$

DYN-EVAL-LOCK
$$\frac{\begin{array}{c} V(x) = \iota \qquad H(\iota) = (C, F, \textbf{unlocked}) \qquad \iota \notin \mathcal{L} \\ H' = H[\iota \mapsto (C, F, \textbf{locked})] \qquad \mathcal{L}' \equiv \mathcal{L} \cup \{\iota\} \end{array}}{\langle H; V; (\mathcal{L}, \textbf{lock}(x) \textbf{ in } e)\rangle \hookrightarrow \langle H'; V; (\mathcal{L}', \textbf{locked}_\iota\{e\})\rangle}$$

DYN-EVAL-LOCK-REENTRANT
$$\frac{V(x) = \iota \qquad H(\iota) = (C, F, \textbf{locked}) \qquad \iota \in \mathcal{L}}{\langle H; V; (\mathcal{L}, \textbf{lock}(x) \textbf{ in } e)\rangle \hookrightarrow \langle H; V; (\mathcal{L}, e)\rangle}$$

DYN-EVAL-LOCK-RELEASE
$$\frac{\begin{array}{c} H(\iota) = (C, F, \textbf{locked}) \qquad \mathcal{L}' \equiv \mathcal{L} \setminus \{\iota\} \\ H' = H[\iota \mapsto (C, F, \textbf{unlocked})] \end{array}}{\langle H; V; (\mathcal{L}, \textbf{locked}_\iota\{v\})\rangle \hookrightarrow \langle H'; V; (\mathcal{L}', v)\rangle}$$

Evaluation Contexts

- A trick to abstract over congruence rules for dynamic semantics

$$\frac{\langle H; V; e \rangle \hookrightarrow \langle H'; V'; e' \rangle}{\langle H; V; x\,.f = e \rangle \hookrightarrow \langle H'; V'; x\,.f = e' \rangle}$$

$$\frac{\langle H; V; e \rangle \hookrightarrow \langle H'; V'; e' \rangle}{\langle H; V; x\,.\,m(e) \rangle \hookrightarrow \langle H'; V'; x\,.\,m(e') \rangle}$$

$$\frac{\langle H; V; e \rangle \hookrightarrow \langle H'; V'; e' \rangle}{\langle H; V; \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \rangle \hookrightarrow \langle H'; V'; \textbf{let}\ x = e_1'\ \textbf{in}\ e_2 \rangle}$$

Evaluation Contexts

- A trick to abstract over congruence rules for dynamic semantics
- Introduce a single rule with an external structure, the *evaluation context*

$$E[\,\bullet\,] ::= x\,.f = \bullet \mid x\,.m(\,\bullet\,) \mid \textbf{let}\ x = \bullet\ \textbf{in}\ e$$

$$\frac{\langle H; V; e\rangle \hookrightarrow \langle H'; V'; e'\rangle}{\langle H; V; E[e]\rangle \hookrightarrow \langle H'; V'; E[e']\rangle}$$

Evaluation Contexts i OOlong

$$E[\bullet] ::= x.f = \bullet \mid x.m(\bullet) \mid \textbf{let } x = \bullet \textbf{ in } e \mid (t)\ \bullet \mid \textbf{locked}_\iota\{\bullet\}$$

$\boxed{cfg_1 \hookrightarrow cfg_2}$ *(Evaluation of expressions)*

DYN-EVAL-CONTEXT

$$\frac{\langle H; V; (\mathcal{L}, e)\rangle \hookrightarrow \langle H'; V'; (\mathcal{L}', e')\rangle}{\langle H; V; (\mathcal{L}, E[e])\rangle \hookrightarrow \langle H'; V'; (\mathcal{L}', E[e'])\rangle}$$

Evaluation Contexts in Coq

- Evaluation contexts can be modelled as functions

```
Definition econtext := expr -> expr.
Definition ctx_call (x : var) (m : method_id) : econtext := fun e => ECall x m e.
```

- An inductive type lists all possible evaluation contexts

```
Inductive is_econtext : econtext -> Prop :=
  | EC_Call : forall x m, is_econtext (ctx_call x m)
```

A rule in the dynamic semantics

```
| EvalContext :
    forall H H' V V' n n' ctx e e' Ls Ls',
      is_econtext ctx ->
      P / (H,  V,  n,  T_Thread Ls  e) ==>
          (H', V', n', T_Thread Ls' e') ->
      P / (H,  V,  n,  T_Thread Ls  (ctx e)) ==>
          (H', V', n', T_Thread Ls' (ctx e'))
```

OOlong — Concurrency

$\boxed{cfg_1 \hookrightarrow cfg_2}$ *(Concurrency)*

DYN-EVAL-ASYNC-LEFT

$$\frac{\langle H; V; T_1 \rangle \hookrightarrow \langle H'; V'; T_1' \rangle}{\langle H; V; T_1 \,||\, T_2 \rhd e \rangle \hookrightarrow \langle H'; V'; T_1' \,||\, T_2 \rhd e \rangle}$$

DYN-EVAL-ASYNC-RIGHT

$$\frac{\langle H; V; T_2 \rangle \hookrightarrow \langle H'; V'; T_2' \rangle}{\langle H; V; T_1 \,||\, T_2 \rhd e \rangle \hookrightarrow \langle H'; V'; T_1 \,||\, T_2' \rhd e \rangle}$$

DYN-EVAL-SPAWN

$$\frac{e = \textbf{finish}\ \{\ \textbf{async}\ \{\ e_1\ \}\ \textbf{async}\ \{\ e_2\ \}\ \}\ ; e_3}{\langle H; V; (\mathcal{L}, e) \rangle \hookrightarrow \langle H; V; (\mathcal{L}, e_1) \,||(\emptyset, e_2) \rhd e_3 \rangle}$$

DYN-EVAL-SPAWN-CONTEXT

$$\frac{\langle H; V; (\mathcal{L}, e) \rangle \hookrightarrow \langle H; V; (\mathcal{L}, e_1) \,||(\emptyset, e_2) \rhd e_3 \rangle}{\langle H; V; (\mathcal{L}, E[e]) \rangle \hookrightarrow \langle H; V; (\mathcal{L}, e_1) \,||(\emptyset, e_2) \rhd E[e_3] \rangle}$$

DYN-EVAL-ASYNC-JOIN

$$\frac{}{\langle H; V; (\mathcal{L}, v) \,||(\mathcal{L}', v') \rhd e \rangle \hookrightarrow \langle H; V; (\mathcal{L}, e) \rangle}$$

Well-Formedness Rules

$\boxed{\Gamma \vdash \langle H; V; T \rangle : t}$ *(Well-formed configuration)*

WF-CFG

$$\frac{\Gamma \vdash H \qquad \Gamma \vdash V \qquad \Gamma \vdash T : t \qquad H \vdash_{\text{lock}} T}{\Gamma \vdash \langle H; V; T \rangle : t}$$

WF-HEAP

$$\frac{\forall \iota : C \in \Gamma.H(\iota) = (C, F, L) \land \Gamma; C \vdash F \qquad \forall \iota \in \mathbf{dom}\,(H).\iota \in \mathbf{dom}(\Gamma) \qquad \vdash \Gamma}{\Gamma \vdash H}$$

WF-T-ASYNC

$$\frac{\Gamma \vdash T_1 : t_1 \qquad \Gamma \vdash T_2 : t_2 \qquad \Gamma \vdash e : t}{\Gamma \vdash T_1 \,||\, T_2 \rhd e : t}$$

WF-T-THREAD

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash (\mathcal{L}, e) : t}$$

WF-T-EXN

$$\frac{\vdash t \qquad \vdash \Gamma}{\Gamma \vdash \mathbf{EXN} : t}$$

WF-FIELDS

$$\frac{\mathbf{fields}\,(C) \equiv f_1 : t_1, \ldots, f_n : t_n \qquad \Gamma \vdash v_1 : t_1, \ldots, \Gamma \vdash v_n : t_n}{\Gamma; C \vdash f_1 \mapsto v_1, \ldots, f_n \mapsto v_n}$$

WF-L-THREAD

$$\frac{\forall \iota \in \mathcal{L}.H(\iota) = (C, F, \mathbf{locked}) \qquad \mathit{distinctLocks}(e) \qquad \forall \iota \in \mathbf{locks}\,(e).\iota \in \mathcal{L}}{H \vdash_{\text{lock}} (\mathcal{L}, e)}$$

WF-VARS

$$\frac{\forall x : t \in \Gamma.V(x) = v \land \Gamma \vdash v : t \qquad \forall x \in \mathbf{dom}\,(V).x \in \mathbf{dom}(\Gamma) \qquad \vdash \Gamma}{\Gamma \vdash V}$$

WF-L-ASYNC

$$\frac{\mathbf{heldLocks}\,(T_1) \cap \mathbf{heldLocks}\,(T_2) \equiv \emptyset \qquad \forall \iota \in \mathbf{locks}\,(e).\iota \in \mathbf{heldLocks}\,(T_1) \qquad \mathit{distinctLocks}(e) \qquad H \vdash_{\text{lock}} T_1 \qquad H \vdash_{\text{lock}} T_2}{H \vdash_{\text{lock}} T_1 \,||\, T_2 \rhd e}$$

WF-L-EXN

$$\frac{}{H \vdash_{\text{lock}} \mathbf{EXN}}$$

UPPSALA UNIVERSITET

OOlong — Type Soundness

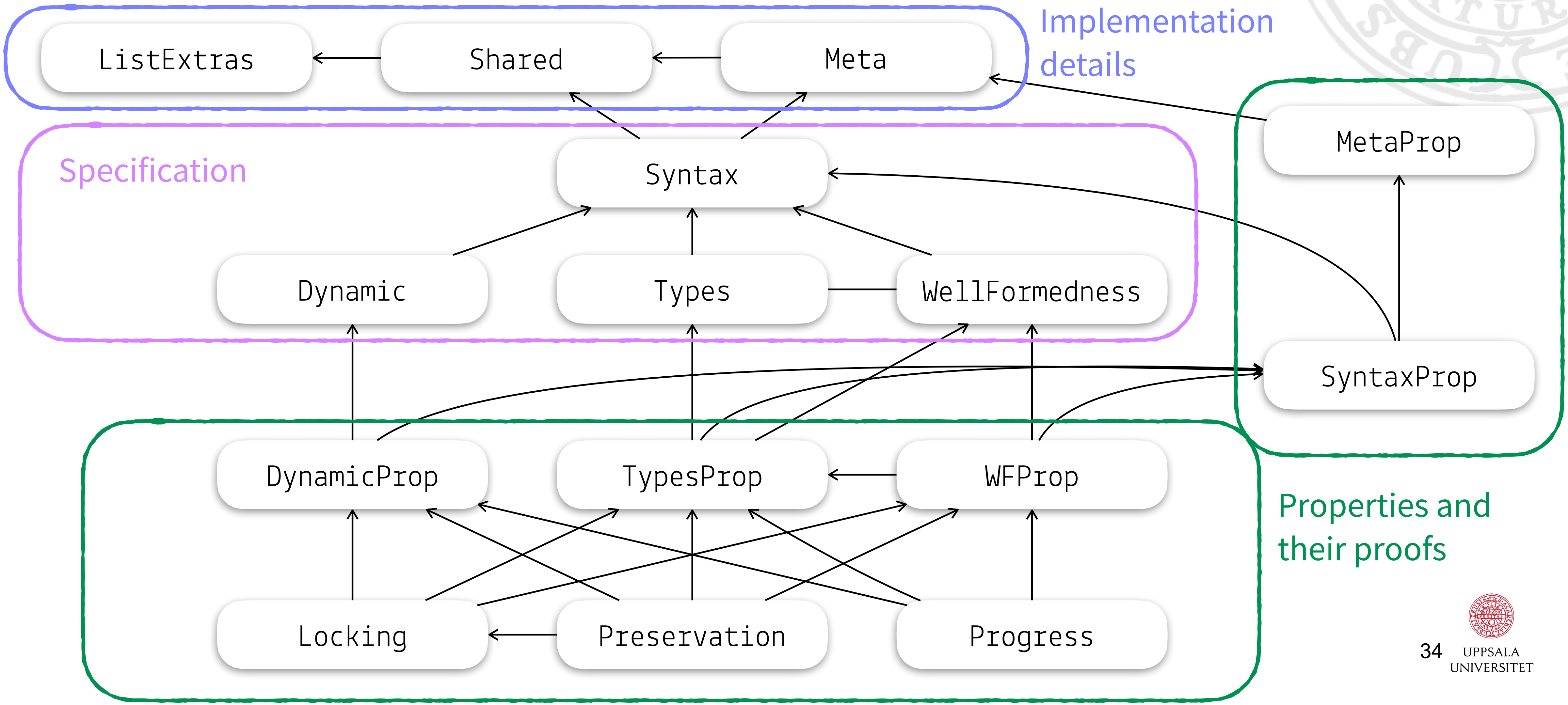
PROGRESS. *A well-formed configuration is either done, has thrown an exception, has deadlocked, or can take one additional step:*

$$\forall \Gamma,\ H,\ V,\ T,\ t\ .\ \Gamma \vdash \langle H; V; T\rangle : t \Rightarrow$$
$$T = (\mathcal{L}, v) \lor T = \textbf{\textit{EXN}} \lor Blocked(\langle H; V; T\rangle) \lor$$
$$\exists cfg', \langle H; V; T\rangle \hookrightarrow cfg'$$

PRESERVATION. *If $\langle H; V; T\rangle$ types to t under some environment Γ, and $\langle H; V; T\rangle$ steps to some $\langle H'; V'; T'\rangle$, there exists an environment subsuming Γ which types $\langle H'; V'; T'\rangle$ to t.*

$$\forall \Gamma,\ H,\ H',\ V,\ V',\ T,\ T',\ t.$$
$$\Gamma \vdash \langle H; V; T\rangle : t \land \langle H; V; T\rangle \hookrightarrow \langle H'; V'; T'\rangle \Rightarrow$$
$$\exists \Gamma'.\Gamma' \vdash \langle H'; V'; T'\rangle : t \land \Gamma \subseteq \Gamma'$$

Modules in the Mechanisation



Mechanised Semantics

- Total weight: ~4100 LOC
  - Specification: ~1700 LOC
  - Proofs: ~2200 LOC
  - Custom tactics: ~200 LOC

~800 lines proofs about locking

- Uses tactics from TLC (no standard library)
- Used to also rely on Adam Chlipala's `crush` tactic from CPDT

- The mechanisation adds "uninteresting" details (fresh variables etc.)
- There are also differences due to presentation (indexed heap, static expressions, etc.)

Comparison of Mechanisations

	FJ	ClJ	ConJ	MJ	LJ	WJ	**OOlong**
State		×	×	×	×	×	×
Statements				×	×	×	
Expressions	×	×	×	×			×
Class Inheritance	×	×	×	×	×	×	
Interfaces		×					×
Concurrency			×			×	×
Stack				×		×	
Mechanised	×*				×		×
$\LaTeX$ sources					×	×	×

~2600 LOC

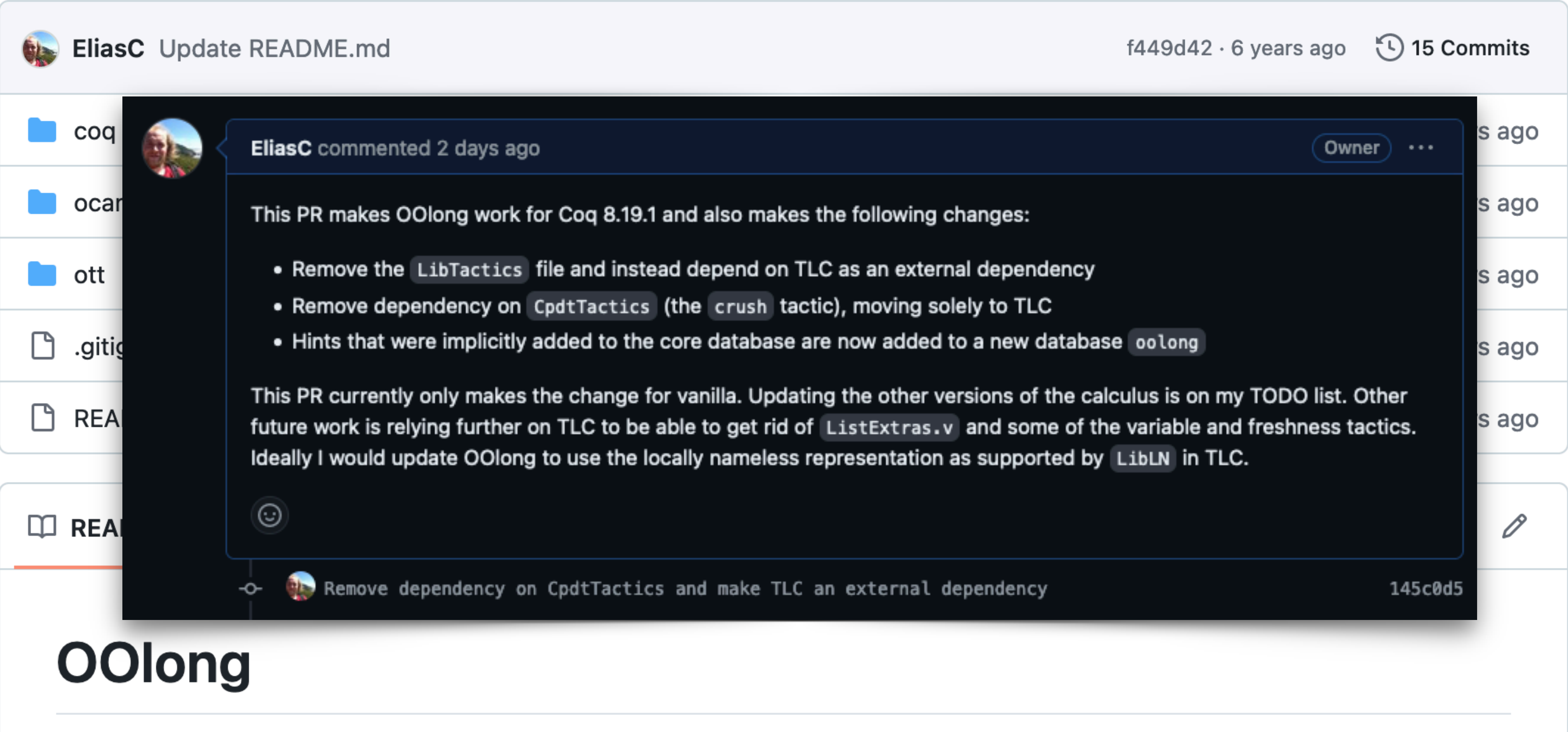
~2300 LOC

~6500 LOC

~4100 LOC

Lets look at some code!

OOlong Sources Available Online



Conclusions

- My first real Coq project was a bit rough around the edges
- There are things I would do differently today
  - Reinvent fewer wheels (lists, maps, environments, variables…)
  - Move smarter, not faster, and thereby get there sooner
- The most important thing I learned was *proof engineering*
- I highly recommend developing your next formalism in Coq!

Other proof assistants are also cool!

Best of Cluck in the Future!

Image generated by openart.ai

UPPSALA UNIVERSITET