OOlong: An Extensible Concurrent Object Calculus

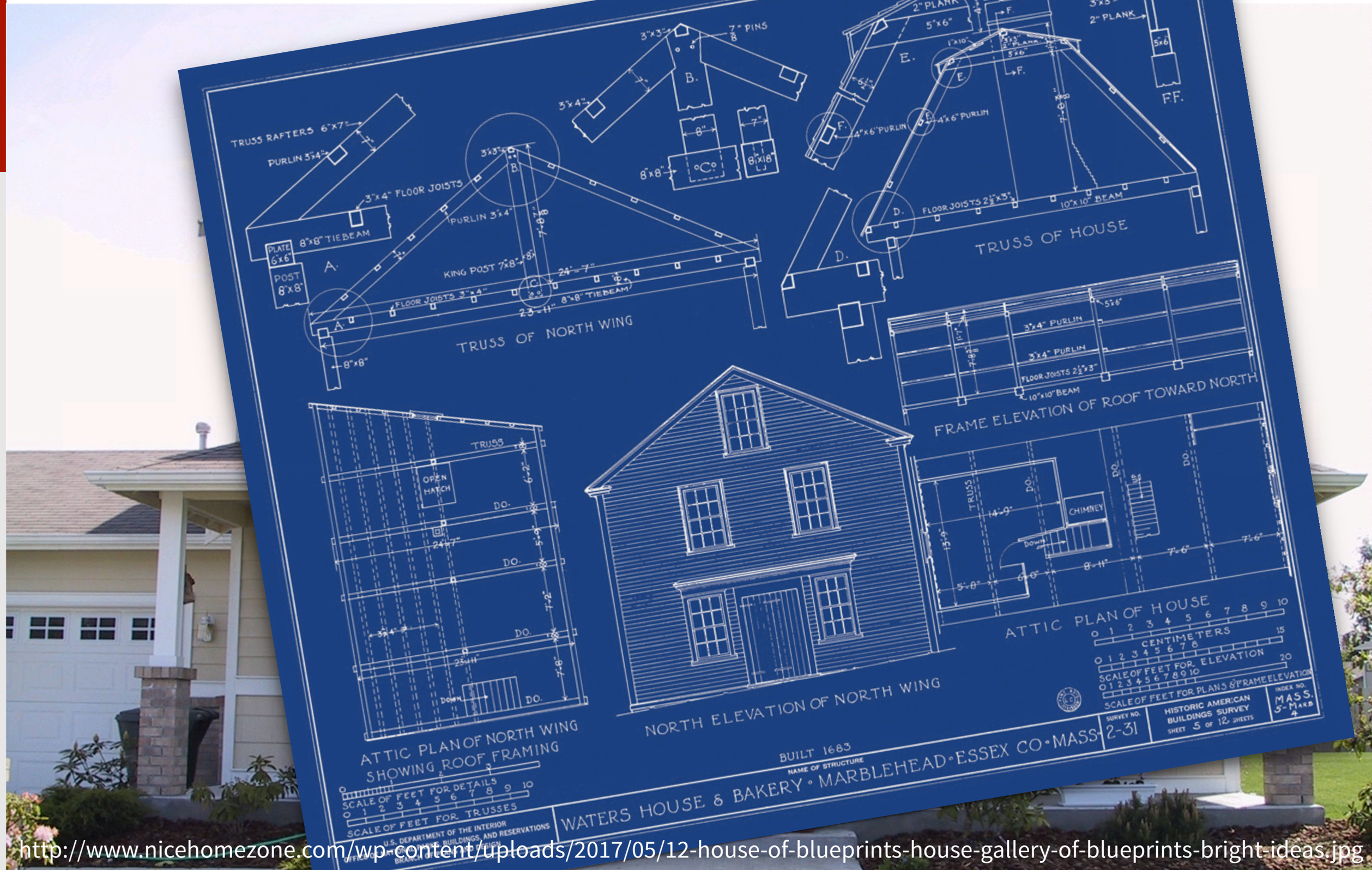
Elias Castegren Tobias Wrigstad
Uppsala University, Sweden

OOPPS@SAC'18
Pau, France

Modelling Reality

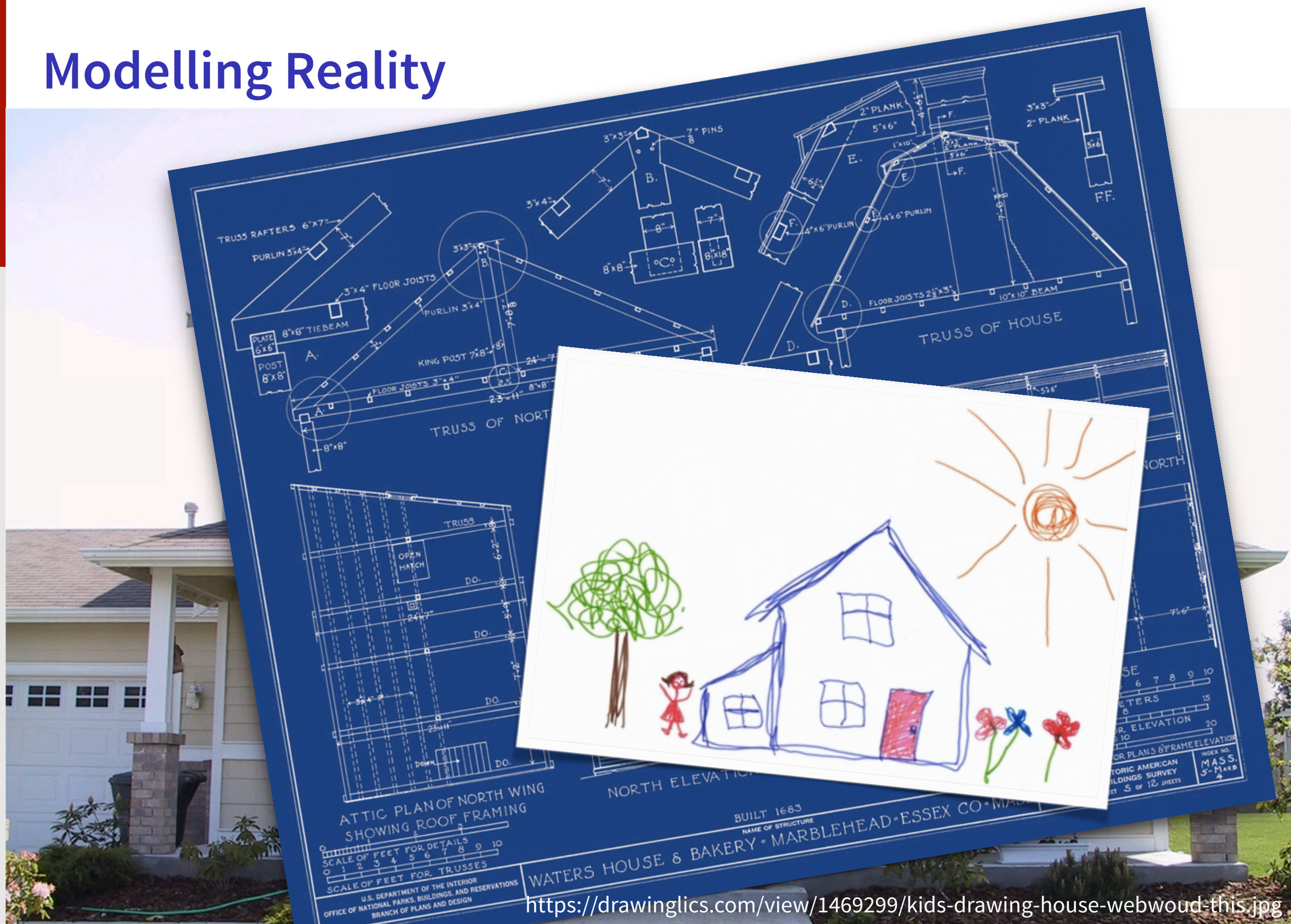
https://s3-us-west-2.amazonaws.com/sah-live/999/CE6Mzya96CvUNevbqtVCysv36np-JFX8.jpg

Modelling Reality



http://www.nicehomezone.com/wp-content/uploads/2017/05/12-house-of-blueprints-house-gallery-of-blueprints-bright-ideas.jpg

Modelling Reality



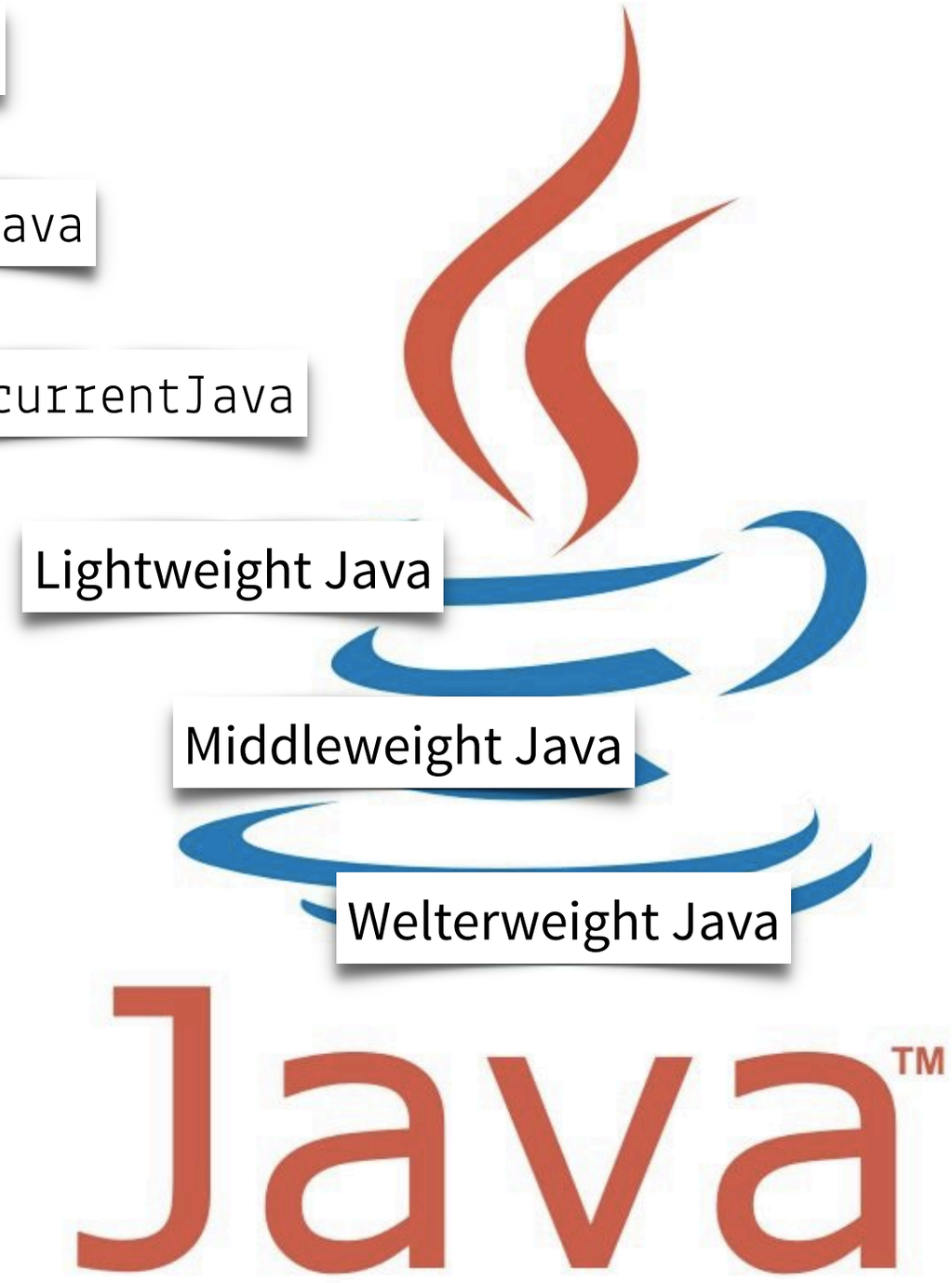
https://drawinglics.com/view/1469299/kids-drawing-house-webwoud-this.jpg

Modelling Programming Languages

Featherweight Java

ClassicJava

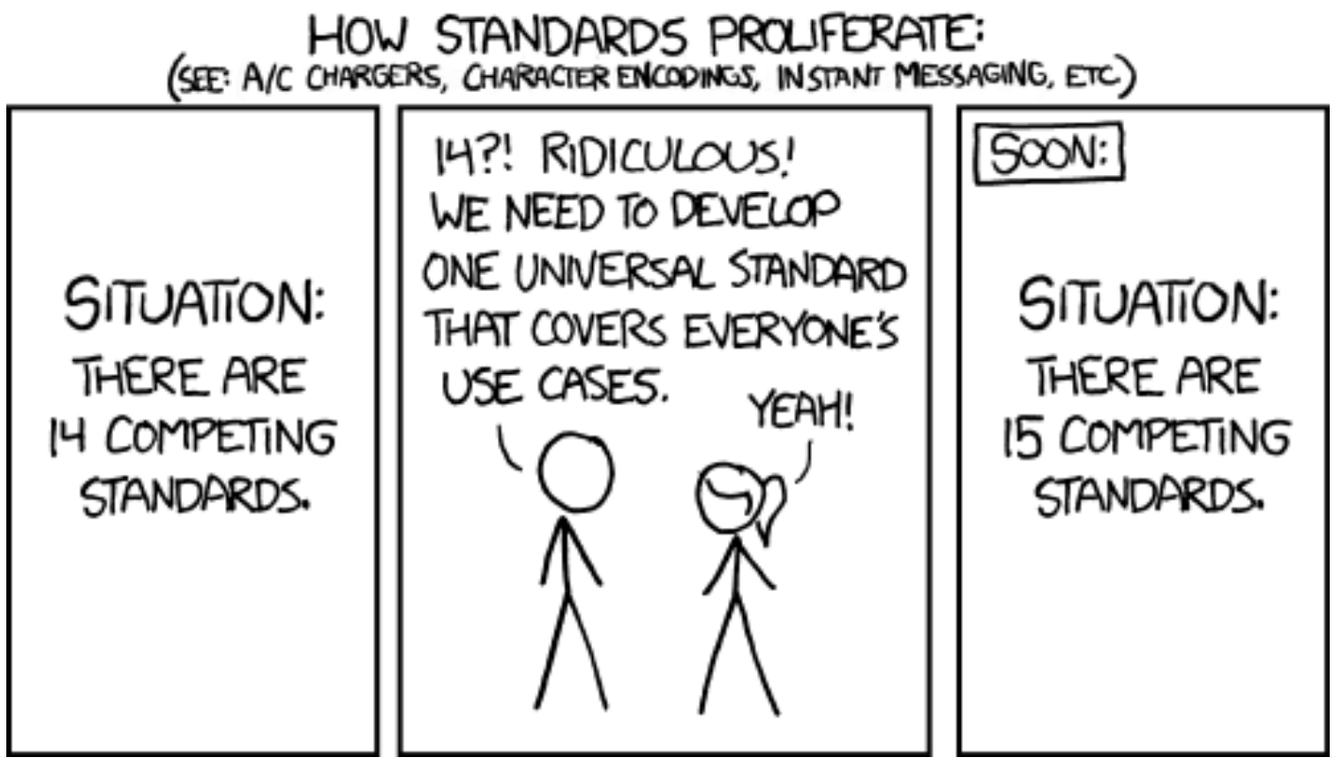
ConcurrentJava



Lightweight Java

Middleweight Java

Welterweight Java



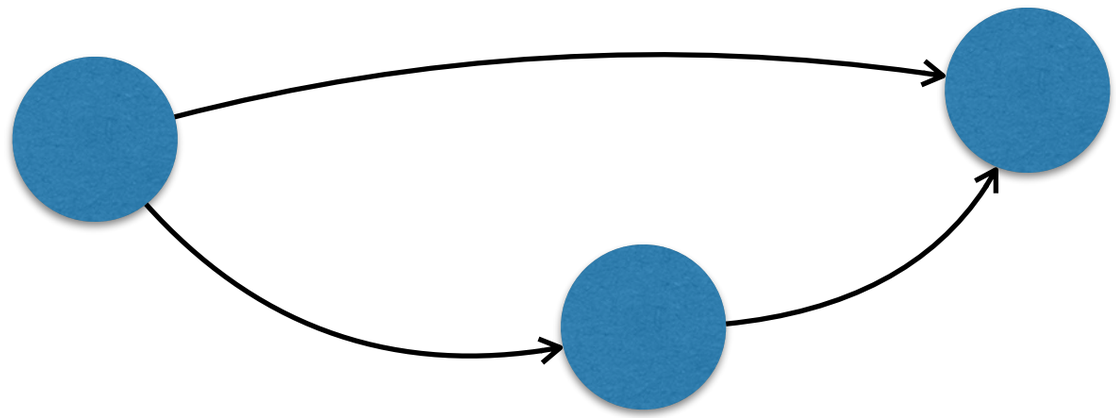
https://imgs.xkcd.com/comics/standards.png

Different Calculi have Different Level of Detail

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

Key Features of Object-Orientation?

- Aliasing and mutable state



- Subtype polymorphism — Not necessarily inheritance!

```
public static void foo(List<Integer> l) {
  l.append(42);
}

public static void main(String args[]) {
  ArrayList<Integer> l = new ArrayList<Integer>();
  foo(l);
  return 0;
}
```

OOlong — An Extensible Object Calculus

- **Design goals**
  - Aliasing and state
  - Subtyping
  - Concurrency and synchronisation
  - Mechanised semantics
  - Simple to reuse and extend

https://www.vizcares.com/wp-content/uploads/2017/07/4-6-450x300.jpg

OOlong is not Java

- Aims to model object-oriented languages in general
- Not tied to Java's design

≠

https://www.zizira.com/wp-content/uploads/2016/06/Pure-Oolong-Tea-Earthy-Strong-and-Rich.png

http://www.deenafarms.com/wp-content/uploads/2017/10/Coffee-Beans.jpg

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 — Comparison

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

OOlong — Syntax

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

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

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

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

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

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

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

OOlong — Type Soundness

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

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

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.*

$$\begin{aligned}\forall \Gamma,\ H,\ H',\ V,\ V',\ T,\ T',\ t. \\ \Gamma \vdash \langle H; V; T\rangle : t \wedge \langle H; V; T\rangle \hookrightarrow \langle H'; V'; T'\rangle \Rightarrow \\ \exists \Gamma'.\Gamma' \vdash \langle H'; V'; T'\rangle : t \wedge \Gamma \subseteq \Gamma'\end{aligned}$$

Mechanised Semantics

- Fully mechanised in Coq
  + ”uninteresting” details
    (fresh variables etc.)
- Follows style of *Software Foundations* (Pierce et al.)
- Total weight: ~4100 LOC
  - Specification: ~1700 LOC
  - Proofs: ~2200 LOC
  - Tactics etc: ~200 LOC
- Also makes use of LibTactics and Adam Chlipala’s `crush` tactic

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

$$\textsc{WF-VAR}\ \frac{\vdash \Gamma \qquad \Gamma(x) = t}{\Gamma \vdash x : t} \qquad \textsc{WF-LET}\ \frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma, x : t_1 \vdash e_2 : t}{\Gamma \vdash \mathbf{let}\, x = e_1\, \mathbf{in}\, e_2 : t}$$

```
Inductive hasType (P : program) (Gamma : env) :
                  expr -> ty -> Prop :=
  | T_Var :
      forall x t,
        wfEnv P Gamma ->
        Gamma (env_var x) = Some t ->
        P; Gamma |- EVar x \in t
  | T_Let :
      forall x e1 e2 t1 t,
        P; Gamma |- e1 \in t1 ->
        P; extend Gamma (env_var (SV x)) t1 |-
               e2 \in t ->
        no_locks e2 ->
        P; Gamma |- ELet x e1 e2 \in t
...
```

Comparison of Mechanisations

~2600 LOC

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

~2300 LOC

~6500 LOC

~4100 LOC

Typesetting OOlong with OTT

- Ott is a tool for writing language definitions [Zappa Nardelli et al.]
- Syntax checking of type-rules
- Generates LaTeX figures

```
G |- e1 : t1
G, x : t1 |- e2 : t
-------------------------- :: let
G |- let x = e1 in e2 : t
```

WF-LET

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

OOlong Sources Available Online

---

6 commits | 1 branch | 0 releases | 1 contributor

Branch: master ▾ | New pull request | Create new file | Upload files | Find file | Clone or download ▾

EliasC Better naming for blocked — Latest commit b86b30b on 1 Dec 2017

coq	Initial commit	4 months ago
ott	Better naming for blocked	4 months ago
.gitignore	Initial commit	4 months ago
README.md	Headers are nice	4 months ago

README.md

OOlong

This repository contains the Ott and Coq sources for OOlong, a concurrent object calculus for modelling object-oriented languages. The calculus was first presented in the article "OOlong: An Extensible Concurrent Object Calculus" (link forthcoming).

Conclusion

- OOlong is a simple and extensible object calculus
  - Aliasing and state
  - Subtyping (no inheritance)
  - Concurrency and synchronisation
- Mechanised semantics for rigorous formalisation
- OTT sources for reusability and LaTeX typesetting
- Sources publicly available for researchers looking to avoid reinventing the wheel

Conclusion

- OOlong is a simple and extensible object calculus
  - Aliasing and state
  - Subtyping (no inheritance)
  - Concurrency and synchronisation
- Mechanised semantics for rigorous formalisation
- OTT sources for reusability and LaTeX typesetting
- Sources publicly available for researchers looking to avoid reinventing the wheel

Thank you!

https://www.vizcares.com/wp-content/uploads/2017/07/4-6-450x300.jpg