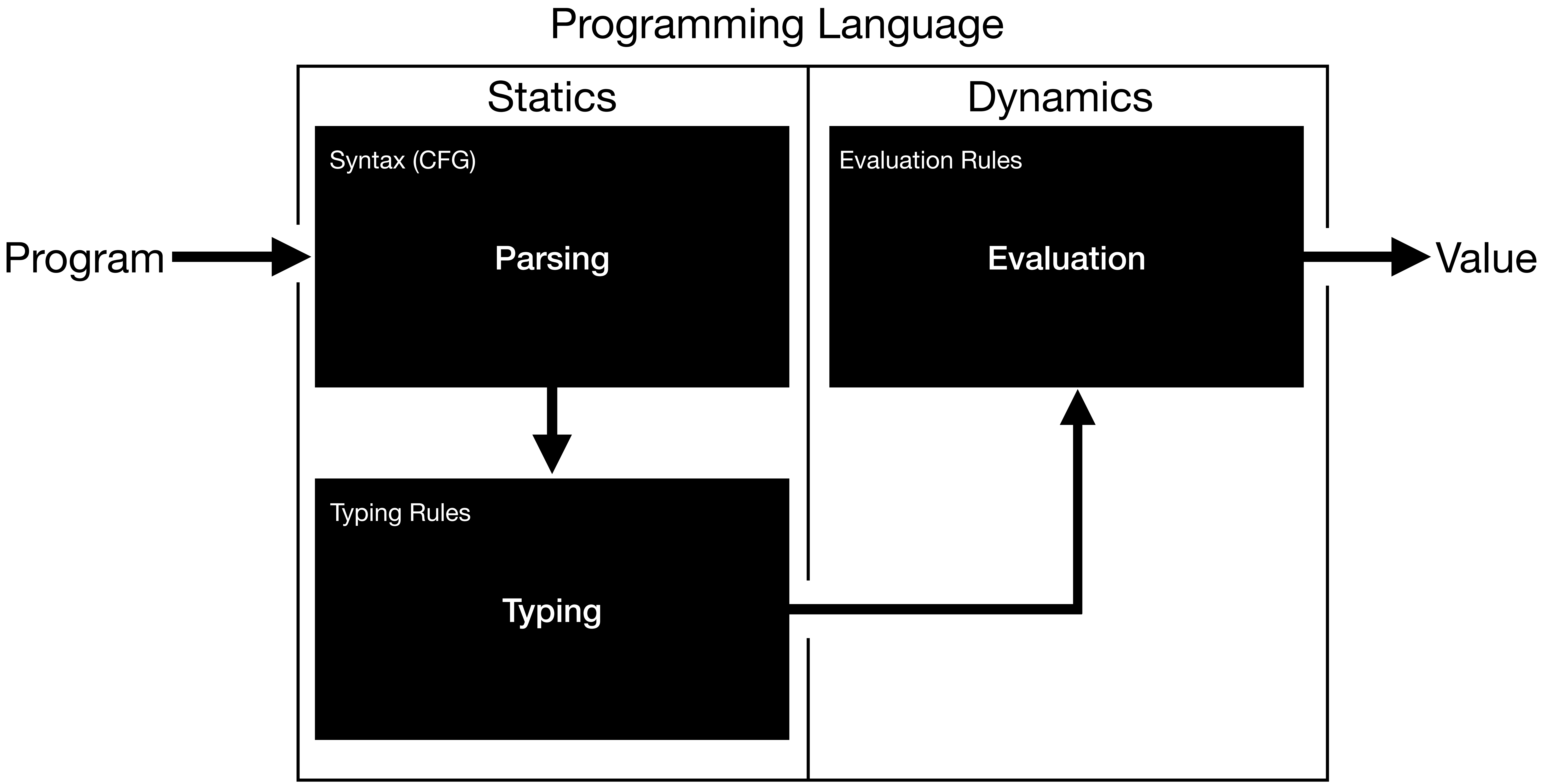
Object-Oriented Programming

Harley Eades III


Programming Language
Statics
Dynamics
Syntax (CFG)
Parsing
Evaluation Rules
Evaluation
Program
Value
Typing Rules
Typing

Part 1: Base System

Base Syntax

Patterns

$p ::= x$
$\mid ()$
$\mid \mathsf{T}$
$\mid \mathsf{F}$
$\mid 0$
$\mid \mathsf{succ}(x)$

Nats

$n ::= 0$
$\mid \mathsf{succ}(n)$

Terms

$t ::= \mathsf{T}$
$\mid \mathsf{F}$
$\mid 0$
$\mid \mathsf{succ}(t)$
$\mid x$
$\mid \mathsf{fun}\, x : T \rightarrow \{t\}$
$\mid t_1\, t_2$
$\mid ()$
$\mid \mathsf{match}\, t\ \{ \mid p_1 \rightarrow t_1 \mid \ldots \mid p_i \rightarrow t_i\}$

Values

$v ::= \mathsf{T}$
$\mid \mathsf{F}$
$\mid \mathsf{fun}\, x : T \rightarrow \{t\}$
$\mid ()$
$\mid n$

Types

$T ::= \mathsf{Bool}$
$\mid \mathsf{Nat}$
$\mid ()$
$\mid T_1 \rightarrow T_2$

Statics: Bools

$$\frac{\Gamma \vdash t : \mathsf{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathsf{match}\, t \,\{ \mid \mathsf{T} \to t_2 \mid \mathsf{F} \to t_3 \} : T}\ \text{If}$$

$$\frac{}{\Gamma \vdash \mathsf{T} : \mathsf{Bool}}\ \text{T}$$

$$\frac{}{\Gamma \vdash \mathsf{F} : \mathsf{Bool}}\ \text{F}$$

Statics: Nats

$$\frac{}{\Gamma \vdash 0 : \mathsf{Nat}}\ \text{0} \qquad \frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}(t) : \mathsf{Nat}}\ \text{succ}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat} \qquad \Gamma \vdash t_2 : T \qquad \Gamma, x : \mathsf{Nat} \vdash t_3 : T}{\Gamma \vdash \mathsf{match}\, t \,\{\ |\ \mathsf{0} \to t_2 \mid \mathsf{succ}(x) \to t_3\} : T}\ \text{matchNat}$$

Statics: Functions

$$\frac{}{\Gamma_1, x : T, \Gamma_2 \vdash x : T}\ \text{Var}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \textsf{fun}\, x : T_1 \to \{t\} : T_1 \to T_2}\ \text{Fun}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1\, t_2 : T_2}\ \text{App}$$

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \textsf{match}\, t_1 \;\{ \mid x \to t_2 \} : T}\ \text{Let}$$

Statics: Unit

$$\frac{}{\Gamma \vdash () : ()}\ \text{Unit}$$

$$\frac{\Gamma \vdash t_1 : () \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash \mathsf{match}\ t_1\ \{\ |\ () \to t_2\} : T}\ \text{Match}$$

Call-by-Value Dynamics: Match

$$\frac{}{\textsf{match}\ v\ \{\ |\ x \to t\} \rightsquigarrow [v/x]t}\ \textsf{let}\beta$$

$$\frac{}{\textsf{match T}\ \{\ |\ \textsf{T} \to t_1\ |\ \textsf{F} \to t_2\} \rightsquigarrow t_1}\ \textsf{if}\beta_1$$

$$\frac{}{\textsf{match}\ ()\ \{\ |\ () \to t\} \rightsquigarrow t}\ \textsf{unit}\beta$$

$$\frac{}{\textsf{match F}\ \{\ |\ \textsf{T} \to t_1\ |\ \textsf{F} \to t_2\} \rightsquigarrow t_2}\ \textsf{if}\beta_2$$

$$\frac{}{\textsf{match}\ 0\ \{\ |\ 0 \to t_1\ |\ \textsf{succ}(x) \to t_2\} \rightsquigarrow t_1}\ \textsf{nat}\beta_1$$

$$\frac{}{\textsf{match succ}(n)\ \{\ |\ 0 \to t_1\ |\ \textsf{succ}(x) \to t_2\} \rightsquigarrow [n/x]t_2}\ \textsf{nat}\beta_2$$

Call-by-Value Dynamics: Match

$$\frac{t \rightsquigarrow t'}{\textsf{match}\ t\ \{\ \mid p_1 \to t_1 \mid \ldots \mid p_i \to t_i\} \rightsquigarrow \textsf{match}\ t'\ \{\ \mid p_1 \to t_1 \mid \ldots \mid p_i \to t_i\}}\ \text{match}$$

Call-by-Value Dynamics: Functions

$$\frac{t_1 \rightsquigarrow t_1'}{t_1\, t_2 \rightsquigarrow t_1'\, t_2}\ \text{App1}$$

$$\frac{t_2 \rightsquigarrow t_2'}{v_1\, t_2 \rightsquigarrow v_1\, t_2'}\ \text{App2}$$

$$\frac{}{(\mathsf{fun}\, x : T \to \{t\})\, v \rightsquigarrow [v/x]t}\ \beta$$

Part 2:Unit

Example: Side Effects

$$\frac{\Gamma \vdash t : \mathsf{String}}{\Gamma \vdash \mathsf{print}\, t : ()}\ \text{Print}$$

Outputting a string to the screen doesn't return a value, and so we can model this by returning the unit.

Statics: Unit

$$\frac{}{\Gamma \vdash () : ()}\ \text{Unit}$$

$$\frac{\Gamma \vdash t_1 : () \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash \mathsf{match}\ t_1 \{\ |\ () \to t_2\} : T}\ \text{Match}$$

Sequencing

$$\frac{\Gamma \vdash t_1 : () \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1; t_2 : T} \text{ Seq}$$

$\text{let } r = \text{ref } 7$

$r := \text{succ}(\,!r); \,!r$

$\#\, 8 : \text{Nat}$

Sequencing

$$\frac{\Gamma \vdash t_1 : () \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1; t_2 : T}\ \text{Seq}$$

$$t_1; t_2 = \mathsf{match}\ t_1\ \{\ |\ () \rightarrow t_2\}$$

$\mathsf{let}\ r = \mathsf{ref}\ 7$

$r := \mathsf{succ}(!r); !r$

$\#\ 8 : \mathsf{Nat}$

Sequencing is match!

Call-by-Value Dynamics: Unit

$$\frac{}{\mathsf{match}\ ()\{\ \mid () \to t_2\} \rightsquigarrow t_2}\ \text{Unit}\beta$$

$$\frac{t_1 \rightsquigarrow t_1'}{\mathsf{match}\ t_1\ \{\ \mid () \to t_2\} \rightsquigarrow \mathsf{match}\ t_1'\ \{\ \mid () \to t_2\}}\ \text{Match}$$

Part 3:Pairs

New Syntactic Forms: Adding Pairs

Terms

$$
\begin{aligned}
t ::= {} & \mathsf{T} \\
\mid {} & \mathsf{F} \\
\mid {} & \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \\
\mid {} & x \\
\mid {} & \text{fun } x : T \to \{t\} \\
\mid {} & t_1\ t_2 \\
\mid {} & (t_1, t_2) \\
\mid {} & t.1 \\
\mid {} & t.2 \\
\mid {} & \text{let } x = t_1 \text{ in } t_2
\end{aligned}
$$

Values

$$
\begin{aligned}
v ::= {} & \mathsf{T} \\
\mid {} & \mathsf{F} \\
\mid {} & \text{fun } x : T \to \{t\} \\
\mid {} & (v_1, v_2)
\end{aligned}
$$

Types

$$
\begin{aligned}
T ::= {} & \text{Bool} \\
\mid {} & (T_1, T_2) \\
\mid {} & T_1 \to T_2
\end{aligned}
$$

Example Programs

$$\text{let twist} = \text{fun}\, p : (\text{Bool}, \text{Bool}) \rightarrow \{(p.2, p.1)\}$$
$$\text{in twist}\,(\text{T}, \text{F})$$

$$\text{let second} = \text{fun}\, p : (\text{Bool} \rightarrow \text{Bool}, \text{Bool}) \rightarrow \{\text{if}\, p.2\, \text{then}\, p.1\, \text{T else F}\}$$
$$\text{in second}\,(\text{fun}\, x : \text{Bool} \rightarrow \{\text{if}\, x\, \text{then F else T}\}, \text{T})$$

Statics: Bools

$$\frac{\Gamma \vdash t_1 : \mathsf{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 : T}\ \text{If}$$

$$\frac{}{\Gamma \vdash \mathsf{T} : \mathsf{Bool}}\ \text{T}$$

$$\frac{}{\Gamma \vdash \mathsf{F} : \mathsf{Bool}}\ \text{F}$$

Statics: Functions

$$\frac{}{\Gamma_1, x : T, \Gamma_2 \vdash x : T}\ \text{Var}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \mathsf{fun}\, x : T_1 \to \{t\} : T_1 \to T_2}\ \text{Fun}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\, t_2 : T_2}\ \text{App}$$

$$\frac{\Gamma \vdash t_1 : T_1 \quad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \mathsf{let}\, x = t_1\, \mathsf{in}\, t_2 : T_2}\ \text{Let}$$

Statics: Pairs

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash (t_1, t_2) : (T_1, T_2)}\ \text{Pair}$$

$$\frac{\Gamma \vdash t : (T_1, T_2)}{\Gamma \vdash t.1 : T_1}\ \text{First}$$

$$\frac{\Gamma \vdash t : (T_1, T_2)}{\Gamma \vdash t.2 : T_2}\ \text{Second}$$

Call-by-Value Dynamics: Bools

$$\frac{t_1 \rightsquigarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightsquigarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3}\ \text{If}$$

$$\frac{}{\text{if T then } t_2 \text{ else } t_3 \rightsquigarrow t_2}\ \text{IfT}$$

$$\frac{}{\text{if F then } t_2 \text{ else } t_3 \rightsquigarrow t_3}\ \text{IfF}$$

Call-by-Value Dynamics: Functions

$$\frac{t_1 \rightsquigarrow t_1'}{t_1\, t_2 \rightsquigarrow t_1'\, t_2}\ \text{App1}$$

$$\frac{}{\mathsf{let}\ x = v\ \mathsf{in}\ t \rightsquigarrow [v/x]t}\ \text{Let}\beta$$

$$\frac{t_2 \rightsquigarrow t_2'}{v_1\, t_2 \rightsquigarrow v_1\, t_2'}\ \text{App2}$$

$$\frac{}{(\mathsf{fun}\ x : T \to \{t\})\ v \rightsquigarrow [v/x]t}\ \beta$$

$$\frac{t_1 \rightsquigarrow t_1'}{\mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \rightsquigarrow \mathsf{let}\ x = t_1'\ \mathsf{in}\ t_2}\ \text{Let}$$

Call-by-Value Dynamics: Pairs

$$\frac{t_1 \rightsquigarrow t_1'}{(t_1, t_2) \rightsquigarrow (t_1', t_2)}\ \text{Pair1} \qquad \frac{t \rightsquigarrow t'}{t.1 \rightsquigarrow t'.1}\ \text{Proj1} \qquad \frac{}{(v_1, v_2).1 \rightsquigarrow v_1}\ \text{Pair}\beta_1$$

$$\frac{t_2 \rightsquigarrow t_2'}{(v_1, t_2) \rightsquigarrow (v_1, t_2')}\ \text{Pair2} \qquad \frac{t \rightsquigarrow t'}{t.2 \rightsquigarrow t'.2}\ \text{Proj2} \qquad \frac{}{(v_1, v_2).2 \rightsquigarrow v_2}\ \text{Pair}\beta_2$$

Part 4:Tuples

New Syntactic Forms: Adding Tuples

Terms

$$
\begin{aligned}
t ::= \ & \mathsf{T} \\
\mid \ & \mathsf{F} \\
\mid \ & \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \\
\mid \ & x \\
\mid \ & \text{fun } x : T \to \{t\} \\
\mid \ & t_1\ t_2 \\
\mid \ & (t_1, \ldots, t_i) \\
\mid \ & \text{match } t_1 \{(x_1, \ldots, x_i) \to t_2\} \\
\mid \ & \text{let } x = t_1 \text{ in } t_2
\end{aligned}
$$

Values

$$
\begin{aligned}
v ::= \ & \mathsf{T} \\
\mid \ & \mathsf{F} \\
\mid \ & \text{fun } x : T \to \{t\} \\
\mid \ & (v_1, \ldots, v_i)
\end{aligned}
$$

Types

$$
\begin{aligned}
T ::= \ & \text{Bool} \\
\mid \ & (T_1, \ldots, T_i) \\
\mid \ & T_1 \to T_2
\end{aligned}
$$

Example: Tuple

$$(\mathsf{T}, \mathsf{F}, \mathsf{T}, \mathsf{F}, \mathsf{F})$$

$$
\begin{array}{l}
\text{fun}\ (p : (\text{Bool}, \text{Bool}, \text{Bool})) \rightarrow \{ \\
\quad\quad \text{match}\ p\ \{ \\
\quad\quad\quad (x, y, z) \rightarrow \begin{array}[t]{l} \text{if}\ x \\ \text{then if}\ y \\ \quad\quad \text{then}\ z \\ \quad\quad \text{else False} \\ \text{else False} \end{array} \\
\quad\quad \} \\
\}
\end{array}
$$

Statics: Bools

$$\frac{\Gamma \vdash t_1 : \text{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T}\ \text{If}$$

$$\frac{}{\Gamma \vdash \text{T} : \text{Bool}}\ \text{T}$$

$$\frac{}{\Gamma \vdash \text{F} : \text{Bool}}\ \text{F}$$

Statics: Functions

$$\frac{}{\Gamma_1, x : T, \Gamma_2 \vdash x : T}\ \text{Var}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \mathsf{fun}\, x : T_1 \to \{t\} : T_1 \to T_2}\ \text{Fun}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1\, t_2 : T_2}\ \text{App}$$

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \mathsf{let}\, x = t_1\, \mathsf{in}\, t_2 : T}\ \text{Let}$$

Statics: Tuples

$$\frac{\Gamma \vdash t_1 : T_1 \quad \cdots \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash (t_1, \ldots, t_i) : (T_1, \ldots, T_i)}\ \text{Tuple}$$

$$\frac{\Gamma \vdash t_1 : (T_1, \ldots, T_i) \qquad \Gamma, x_1 : T_1, \ldots, x_i : T_i \vdash t_2 : T}{\Gamma \vdash \mathsf{match}\ t_1 \{(x_1, \ldots, x_i) \to t_2\} : T}\ \text{Match}$$

Call-by-Value Dynamics: Bools

$$\frac{t_1 \rightsquigarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightsquigarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3}\ \text{If}$$

$$\frac{}{\text{if T then } t_2 \text{ else } t_3 \rightsquigarrow t_2}\ \text{IfT}$$

$$\frac{}{\text{if F then } t_2 \text{ else } t_3 \rightsquigarrow t_3}\ \text{IfF}$$

Call-by-Value Dynamics: Functions

$$\frac{t_1 \rightsquigarrow t_1'}{t_1\, t_2 \rightsquigarrow t_1'\, t_2}\ \text{App1}$$

$$\frac{}{\mathsf{let}\, x = v\, \mathsf{in}\, t \rightsquigarrow [v/x]t}\ \text{Let}\beta$$

$$\frac{t_2 \rightsquigarrow t_2'}{v_1\, t_2 \rightsquigarrow v_1\, t_2'}\ \text{App2}$$

$$\frac{}{(\mathsf{fun}\, x : T \to \{t\})\, v \rightsquigarrow [v/x]t}\ \beta$$

$$\frac{t_1 \rightsquigarrow t_1'}{\mathsf{let}\, x = t_1\, \mathsf{in}\, t_2 \rightsquigarrow \mathsf{let}\, x = t_1'\, \mathsf{in}\, t_2}\ \text{Let}$$

Call-by-Value Dynamics: Tuples

$$\frac{t_{i+1} \rightsquigarrow t'_{i+1}}{(v_1, \ldots, v_i, t_{i+1}, \ldots, t_j) \rightsquigarrow (v_1, \ldots, v_i, t'_{i+1}, \ldots, t_j)} \text{ Tuple}$$

Call-by-Value Dynamics: Tuples

$$\frac{t_1 \rightsquigarrow t_1'}{\mathsf{match}\ t_1 \{(x_1, \ldots, x_i) \to t_2\} \rightsquigarrow \mathsf{match}\ t_1' \{(x_1, \ldots, x_i) \to t_2\}}\ \text{Match}$$

$$\frac{}{\mathsf{match}\ (v_1, \ldots, v_i) \{(x_1, \ldots, x_i) \to t_2\} \rightsquigarrow [v_1/x_1] \cdots [v_i/x_i] t_2}\ \text{Tuple}\beta$$

Part 5:Records

New Syntactic Forms: Adding Records

Suppose we have a set of labels $\mathscr{L}$

Terms

$$
\begin{aligned}
t ::= {} & \mathsf{T} \\
| {} & \mathsf{F} \\
| {} & \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \\
| {} & x \\
| {} & \text{fun } x : T \to \{t\} \\
| {} & t_1\ t_2 \\
| {} & (l_1 = t_1, \ldots, l_i = t_i) \\
| {} & t\,.\,l \\
| {} & \text{let } x = t_1 \text{ in } t_2
\end{aligned}
$$

Values

$$
\begin{aligned}
v ::= {} & \mathsf{T} \\
| {} & \mathsf{F} \\
| {} & \text{fun } x : T \to \{t\} \\
| {} & (l_1 = v_1, \ldots, l_i = v_i)
\end{aligned}
$$

Types

$$
\begin{aligned}
T ::= {} & \text{Bool} \\
| {} & (l_1 : T_1, \ldots, l_i : T_i) \\
| {} & T_1 \to T_2
\end{aligned}
$$

Example: Records

$(x = 2, y = 5) : (x : \mathsf{Int}, y : \mathsf{Int})$

$(\text{desc} = \text{"brake rotor"}, \text{partno} = 3947, \text{cost} = 250) : (\text{desc} : \mathsf{String}, \text{partno} : \mathsf{Int}, \text{cost} : \mathsf{Float})$

Statics: Bools

$$\frac{\Gamma \vdash t_1 : \mathsf{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 : T}\ \textsf{If}$$

$$\frac{}{\Gamma \vdash \mathsf{T} : \mathsf{Bool}}\ \textsf{T}$$

$$\frac{}{\Gamma \vdash \mathsf{F} : \mathsf{Bool}}\ \textsf{F}$$

Statics: Functions

$$\frac{}{\Gamma_1, x : T, \Gamma_2 \vdash x : T}\ \text{Var}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \mathsf{fun}\, x : T_1 \to \{t\} : T_1 \to T_2}\ \text{Fun}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1\, t_2 : T_2}\ \text{App}$$

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \mathsf{let}\, x = t_1\, \mathsf{in}\, t_2 : T}\ \text{Let}$$

Statics: Records

$$\frac{\Gamma \vdash t_1 : T_1 \quad \cdots \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash (l_1 = t_1, \ldots, l_i = t_i) : (l_1 : T_1, \ldots, l_i : T_i)} \text{ Record}$$

$$\frac{\Gamma \vdash t : (l_1 : T_1, \ldots, l_i : T_i)}{\Gamma \vdash t \,.\, l_i : T_i} \text{ Proj}$$

Call-by-Value Dynamics: Bools

$$\frac{t_1 \rightsquigarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightsquigarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3}\ \text{If}$$

$$\frac{}{\text{if T then } t_2 \text{ else } t_3 \rightsquigarrow t_2}\ \text{IfT}$$

$$\frac{}{\text{if F then } t_2 \text{ else } t_3 \rightsquigarrow t_3}\ \text{IfF}$$

Call-by-Value Dynamics: Functions

$$\frac{t_1 \rightsquigarrow t_1'}{t_1\, t_2 \rightsquigarrow t_1'\, t_2}\ \text{App1}$$

$$\frac{}{\mathsf{let}\ x = v\ \mathsf{in}\ t \rightsquigarrow [v/x]t}\ \text{Let}\beta$$

$$\frac{t_2 \rightsquigarrow t_2'}{v_1\, t_2 \rightsquigarrow v_1\, t_2'}\ \text{App2}$$

$$\frac{}{(\mathsf{fun}\ x : T \to \{t\})\ v \rightsquigarrow [v/x]t}\ \beta$$

$$\frac{t_1 \rightsquigarrow t_1'}{\mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \rightsquigarrow \mathsf{let}\ x = t_1'\ \mathsf{in}\ t_2}\ \text{Let}$$

Call-by-Value Dynamics: Records

$$\frac{t_{i+1} \rightsquigarrow t'_{i+1}}{(l_1 = v_1, \ldots, l_i = v_i, l_{i+1} = t_{i+1}, \ldots, l_j = t_j) \rightsquigarrow (l_1 = v_1, \ldots, l_i = v_i, l_{i+1} = t'_{i+1}, \ldots, l_j = t_j)}\ \text{Record}$$

Call-by-Value Dynamics: Records

$$\frac{t \rightsquigarrow t'}{t\,.\,l_i \rightsquigarrow t'\,.\,l_i}\ \text{Proj}$$

$$\frac{}{(l_1 = v_1, \ldots, l_i = v_i)\,.\,l_j \rightsquigarrow v_j}\ \text{Record}\beta$$

Part 6: Mutable References

Up until now, all of the languages we have studied have been **pure**.

Up until now, all of the languages we have studied have been **pure**.

pure: a programming language without **computational effects**.

Up until now, all of the languages we have studied have been **pure**.

pure: a programming language without **computational effects**.

computational effect: programs that interact or modify with the outside world

Computational Effects

- mutable references
- input/output
- networking
- non-local transfers of control
- inter-process synchronization

Computational Effects

- mutable references
- input/output
- networking
- non-local transfers of control
- inter-process synchronization

Key Concepts

- allocation (references)
- assignment operator
- explicit dereferencing
- stores (or heaps)

Allocation

Allocating a reference: $\mathrm{ref}\ 5 : \mathrm{Ref\ Nat}$

Allocation

Allocating a reference: ref 5 : Ref Nat

allocate a new cell

Allocation

Allocating a reference: $\text{ref}\ 5 : \text{Ref Nat}$

initial value

allocate a new cell

Allocation

Allocating a reference: ref 5 : Ref Nat

type of initial value

initial value

allocate a new cell

Assignment

Assignment operator: $r := 7$: Unit

Assignment

Assignment operator: $r := 7$: Unit

a reference

Assignment

Assignment operator: $r := 7$: Unit

new value

a reference

Assignment

Assignment operator: $r := 7 : \mathsf{Unit}$

assignment's type

new value

a reference

Assignment

Assignment operator: $r := 7$: Unit

Example:

$$\text{let}\ r = \text{ref}\ 5$$
$$\#\ r : \text{Ref Nat}$$
$$r := 7$$
$$\#\ \text{unit} : \text{Unit}$$

Dereferencing

Dereferencing operator: $!r : \mathrm{Nat}$

Dereferencing

Dereferencing operator: $!r$: Nat

a reference

Dereferencing

Dereferencing operator: $!r : \mathrm{Nat}$

type of value

a reference

Dereferencing

Dereferencing operator: $!r : \mathrm{Nat}$

Example:

$\mathrm{let}\ r = \mathrm{ref}\ 5$

$\#\ r : \mathrm{Ref\ Nat}$

$!r$

$\#\ 5 : \mathrm{Nat}$

$r := 7$

$\#\ \mathrm{unit} : \mathrm{Unit}$

$!r$

$\#\ 7 : \mathrm{Nat}$

Sequencing

$$\frac{\Gamma \vdash t_1 : () \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1; t_2 : T}\ \text{Seq}$$

Sequencing

$$\frac{\Gamma \vdash t_1 : () \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1; t_2 : T}\ \text{Seq}$$

$\mathsf{let}\ r = \mathsf{ref}\ 7$

$r := \mathsf{succ}(\,!r);\ !r$

$\#\ 8 : \mathsf{Nat}$

Sequencing

$$\frac{\Gamma \vdash t_1 : () \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1; t_2 : T}\ \text{Seq}$$

$\text{let } r = \text{ref } 7$
$r := \text{succ}(!r);$
$r := \text{succ}(!r);$
$r := \text{succ}(!r);$
$r := \text{succ}(!r);$
$!r$
$\# 11 : \text{Nat}$

Stores

- Locations are essentially pointers.
- Stores are sets of mappings from locations to values.
- Store typings are sets of locations with their types.
  - Think of these as "contexts" for stores, but rather than free variables and types we have locations and types.

Stores:

$$\mu ::= \varnothing \mid \mu, l = v$$

Store Typings:

$$\Sigma ::= \varnothing \mid \Sigma, l : T$$

Store Substitution:

$$[t/l]\mu = \begin{cases} \varnothing & \text{if } \mu = \varnothing \\ [t/l]\mu, l' = [t/l]v & \text{if } \mu = (\mu, l' = v) \end{cases}$$

Stores

- Stores will be the states during evaluation.
- Store typings will be used to type locations during typing.

Stores:

$$\mu ::= \varnothing \\ \mid \mu, l = v$$

Store Typings:

$$\Sigma ::= \varnothing \\ \mid \Sigma, l : T$$

Base + Mutable References

Terms

$$t ::= \ldots \mid \text{ref } t \mid\ !t \mid t_1 := t_2 \mid l$$

Values

$$v ::= \ldots \mid l$$

Types

$$T ::= \ldots \mid \text{Ref } T$$

Stores

$$\mu ::= \varnothing \mid \mu, l = v$$

Store Typings

$$\Sigma ::= \varnothing \mid \Sigma, l : T$$

Base + Mutable References

Existing typing rules are updated to the judgment

$$\Gamma \mid \Sigma \vdash t : T$$

existing rules don't change except to carry the Σ along to each premise.

$$\frac{\Sigma(l) = T}{\Gamma \mid \Sigma \vdash l : \mathsf{Ref}\, T}\ \text{loc} \qquad \frac{\Gamma \mid \Sigma \vdash t_1 : \mathsf{Ref}\, T \qquad \Gamma \mid \Sigma \vdash t_2 : T}{\Gamma \vdash t_1 := t_2 : T}\ \text{assign}$$

$$\frac{\Gamma \mid \Sigma \vdash t : T}{\Gamma \vdash \mathsf{ref}\, t : \mathsf{Ref}\, T}\ \text{ref} \qquad \frac{\Gamma \mid \Sigma \vdash t : \mathsf{Ref}\, T}{\Gamma \vdash\ !t : T}\ \text{deref}$$

Base + Mutable References

Existing evaluation rules are updated to the judgment

$$[\mu_1 \mid t_1] \rightsquigarrow [\mu_2 \mid t_2]$$

existing rules don't change except to carry the μ along to each premise; where we replace every $t_1 \rightsquigarrow t_2$ with $[\mu_1 \mid t_1] \rightsquigarrow [\mu_2 \mid t_2]$.

$$\frac{l \notin \mathsf{dom}(\mu)}{[\mu \mid \mathsf{ref}\, v] \rightsquigarrow [\mu, l = v \mid l]}\ \mathsf{ref}\beta$$

$$\frac{[\mu_1 \mid t_1] \rightsquigarrow [\mu_2 \mid t_2]}{[\mu_1 \mid \mathsf{ref}\, t_1] \rightsquigarrow [\mu_2 \mid \mathsf{ref}\, t_2]}\ \mathsf{ref}$$

$$\frac{\mu(l) = v}{[\mu \mid !l] \rightsquigarrow [\mu \mid v]}\ \mathsf{deref}\beta$$

$$\frac{[\mu_1 \mid t_1] \rightsquigarrow [\mu_2 \mid t_2]}{[\mu_1 \mid !t_1] \rightsquigarrow [\mu_2 \mid !t_2]}\ \mathsf{deref}$$

Base + Mutable References

Existing evaluation rules are updated to the judgment

$$[\mu_1 \mid t_1] \rightsquigarrow [\mu_2 \mid t_2]$$

existing rules don't change except to carry the μ along to each premise; where we replace every $t_1 \rightsquigarrow t_2$ with $[\mu_1 \mid t_1] \rightsquigarrow [\mu_2 \mid t_2]$.

$$\frac{}{[\mu \mid l := v] \rightsquigarrow [[v/l]\mu \mid ()]}\ \text{assign}\beta$$

$$\frac{[\mu_1 \mid t_1] \rightsquigarrow [\mu_2 \mid t_1']}{[\mu_1 \mid t_1 := t_2] \rightsquigarrow [\mu_2 \mid t_1' := t_2]}\ \text{assign1}$$

$$\frac{[\mu_1 \mid t_2] \rightsquigarrow [\mu_2 \mid t_2']}{[\mu_1 \mid v_1 := t_2] \rightsquigarrow [\mu_2 \mid v_1 := t_2']}\ \text{assign2}$$

Part 7: Subtyping

System: Base + Records

Motivating Example

What type does this program have?

$$(\mathsf{fun}\,(r : (x : \mathsf{Nat}) \to \{r\,.\,x\})\,(x = 0, y = \mathsf{succ}\,0)$$

Motivating Example

What type does this program have?

$$\frac{(\mathsf{fun}\,(r : (x : \mathsf{Nat}) \to \{r\,.\,x\})}{(x : \mathsf{Nat}) \to \mathsf{Nat}}\;(x = 0, y = \mathsf{succ}\,0)$$

Motivating Example

What type does this program have?

$$\frac{(\mathsf{fun}\,(r : (x : \mathsf{Nat}) \to \{r\,.\,x\})}{(x : \mathsf{Nat}) \to \mathsf{Nat}} \quad \frac{(x = 0, y = \mathsf{succ}\,0)}{(x : \mathsf{Nat}, y : \mathsf{Nat})}$$

Motivating Example

What type does this program have?

$$\frac{(\mathsf{fun}\,(r : (x : \mathsf{Nat}) \to \{r\,.\,x\})}{(x : \mathsf{Nat}) \to \mathsf{Nat}} \quad \frac{(x = 0, y = \mathsf{succ}\,0)}{(x : \mathsf{Nat}, y : \mathsf{Nat})}$$

😵 **it’s not typeable!!**

Motivating Example

What type does this program have?

$$\underbrace{(\mathsf{fun}\,(r : (x : \mathsf{Nat}) \to \{r\,.\,x\})}_{(x\,:\,\mathsf{Nat}) \to \mathsf{Nat}}\ \underbrace{(x = 0, y = \mathsf{succ}\,0)}_{(x\,:\,\mathsf{Nat},\, y\,:\,\mathsf{Nat})}$$

😵 it's not typeable!!

identical

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\, t_2 : T_2}\ \text{App}$$

Motivating Example

What type does this program have?

$$\dfrac{(\mathsf{fun}\,(r : (x : \mathsf{Nat}) \to \{r\,.\,x\})}{(x : \mathsf{Nat}) \to \mathsf{Nat}} \quad \dfrac{(x = 0, y = \mathsf{succ}\,0)}{(x : \mathsf{Nat}, y : \mathsf{Nat})}$$

different

$$\dfrac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1' \qquad T_1 <: T_1'}{\Gamma \vdash t_1\, t_2 : T_2}\,\text{App} \qquad \dfrac{\{l_1' : T_1', \ldots, l_j' : T_j'\} \subseteq \{l_1 : T_1, \ldots, l_i : T_i'\}}{(l_1 : T_1, \ldots, l_i : T_i) <: (l_1' : T_1', \ldots, l_j' : T_j')}\,\text{recSub}$$

Motivating Example

Wh

Subtyping

Increases the set of typeable programs by generalizing the types of the programs that flow into another.

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \quad \Gamma \vdash t_2 : T_1' \qquad T_1 <: T_1'}{\Gamma \vdash t_1\, t_2 : T_2}\ \text{App}$$

$$\frac{: T_1', \ldots, l_j' : T_j'\}}{: T_1', \ldots, l_j' : T_j')}\ \text{recSub}$$

Subtyping

<u>principle of safe substitution</u>

S is a <u>subtype</u> of T, written $S <: T$, means any term of type S can safely be used in a context where a term of type T is expected.

Subtyping

subset semantics

S is a subtype of T, written $S <: T$, every value described by S is also described by T.

Subtyping

<u>subsumption</u>

every element t of T_1 is also an element of T_2

$$\frac{\Gamma \vdash t : T_1 \qquad T_1 <: T_2}{\Gamma \vdash t : T_2}\ \text{sub}$$

Subtyping

$$\frac{\Gamma \vdash (x = 0, y = \mathsf{succ}(0)) : (x : \mathsf{Nat}, y : \mathsf{Nat}) \qquad (x : \mathsf{Nat}, y : \mathsf{Nat}) <: (x : \mathsf{Nat})}{\Gamma \vdash (x = 0, y = \mathsf{succ}(0)) : (x : \mathsf{Nat})}\ \mathsf{sub}$$

Base + Subtyping

The only syntax that changes is the addition of a Top type.

Terms	Values	Types
$t ::= \ldots$	$v ::= \ldots$	$T ::= \ldots$
		$\mid \text{Top}$

Base + Subtyping

<u>Subtyping Rules</u>

$$\frac{}{T <: T}\ \text{Refl} \qquad \frac{T_1 <: T_2 \qquad T_2 <: T_3}{T_1 <: T_3}\ \text{Trans} \qquad \frac{}{T <: \mathsf{Top}}\ \text{Top}$$

$$\frac{T_1' <: T_1 \qquad T_2 <: T_2'}{T_1 \to T_2 <: T_1' \to T_2'}\ \text{Arrow}$$

Base + Records + Subtyping

Subtyping Rules

$$\frac{}{(l_i : T_i)^{i \in \{1 \ldots n+k\}} <: (l_i : T_i)^{i \in \{1 \ldots n\}}}\ \text{RecWidth}$$

$$\frac{\forall i \in \{1 \ldots n\}\ .\ T_i <: T'_i}{(l_i : T_i)^{i \in \{1 \ldots n\}} <: (l_i : T'_i)^{i \in \{1 \ldots n\}}}\ \text{RecDepth}$$

$$\frac{(l_i : T_i)^{i \in \{1 \ldots n\}} \text{ is a permutation of } (l_j : T'_j)^{j \in \{1 \ldots n\}}}{(l_i : T_i)^{i \in \{1 \ldots n\}} <: (l_j : T'_j)^{j \in \{1 \ldots n\}}}\ \text{RecDepth}$$

Base + Subtyping

Typing rules are all the same, except the addition of the subsumption rule.

$$\frac{\Gamma \vdash t : T_1 \qquad T_1 <: T_2}{\Gamma \vdash t : T_2}\ \text{Sub}$$

Part 8: Imperative Objects

System: Base + Records + Mutable References + Subtypes

Part 9: OOP in OCaml

What is OCaml?

Core Design Concepts:

Syntax

An object oriented, imperative, functional programming language.

What is OCaml?

Core Design Concepts:

Syntax

An object oriented, imperative, functional programming language.

OCaml mixes all of these paradigms together.

What is OCaml?

Core Design Concepts:

Syntax

An object oriented, imperative, functional programming language.

OCaml mixes all of these paradigms together.

What is OCaml?

Core Design Concepts:

Syntax

An object oriented, imperative, functional programming language.

OCaml mixes all of these paradigms together.

What is OCaml?

Core Design Concepts:

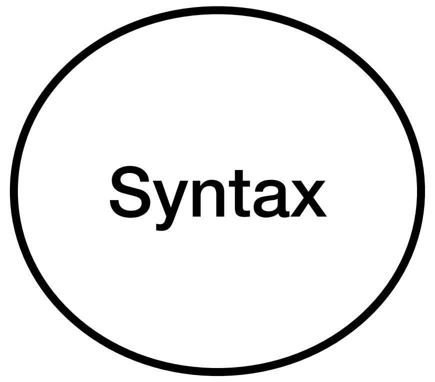
Syntax

An <u>object oriented</u>, imperative, functional programming language.

OCaml mixes all of these paradigms together.

What is OCaml?

Core Design Concepts:



An <u>object oriented</u>, <u>imperative</u>, functional programming language.

OCaml mixes all of these paradigms together.

Class Definitions

Core Design Concepts:

Syntax

```
class name = object (self) … end
```

Class Definitions

Core Design Concepts:

Syntax

```
class stack_of_ints =
  object (self)
    val mutable the_list = ([] : int list)
     …
  end;;
```

Class Definitions

Core Design Concepts:

Syntax

```
class stack_of_ints =
  object (self)
    val mutable the_list = ([] : int list)
    method push x = …
    method pop = …
    method peek = …
    method size = …
  end;;
```

Class Definitions

Core Design Concepts:

Syntax

```
class stack_of_ints =
  object (self)
    val mutable the_list = ([] : int list)
    method push x = the_list <- Cons(x, the_list)
    method pop = …
    method peek = …
    method size = …
  end;;
```

Class Definitions

Core Design Concepts:

Syntax

```
class stack_of_ints =
  object (self)
    val mutable the_list = ([] : int list)
    method pop =
      let result = head the_list in
      the_list <- tail the_list;
      result
    method push x = …
    method peek = …
    method size = …
  end;;
```

Class Definitions

Core Design Concepts:

Syntax

```
class stack_of_ints =
  object (self)
    val mutable the_list = ([] : int list)
    method push x = …
    method pop = …
    method peek = head the_list
    method size = …
  end;;
```

Class Definitions

Core Design Concepts:

Syntax

```
class stack_of_ints =
  object (self)
    val mutable the_list = ([] : int list)
    method push x = …
    method pop = …
    method peek = …
    method size = length the_list
  end;;
```

Class Definitions

Core Design Concepts:

Syntax

```
class stack_of_ints =
  object (self) …
end;;
class stack_of_ints :
  object
    val mutable the_list : int list
    method peek : int
    method pop : int
    method push : int -> unit
    method size : int
  end
```

Accessing fields and methods

Core Design Concepts:

Syntax

```
# let s = new stack_of_ints;;
val s : stack_of_ints = <obj>
```

Accessing fields and methods

Core Design Concepts:

Syntax

```
s#fieldName

s#methodName
```

Accessing fields and methods

Core Design Concepts:

Syntax

```
# for i = 1 to 10 do
    s#push i
  done;;
- : unit = ()
# while s#size > 0 do
    Printf.printf "Popped %d off the stack.\n" s#pop
  done;;
…
Popped 10 off the stack.
Popped 9 off the stack.
Popped 8 off the stack.
- : unit = ()
```