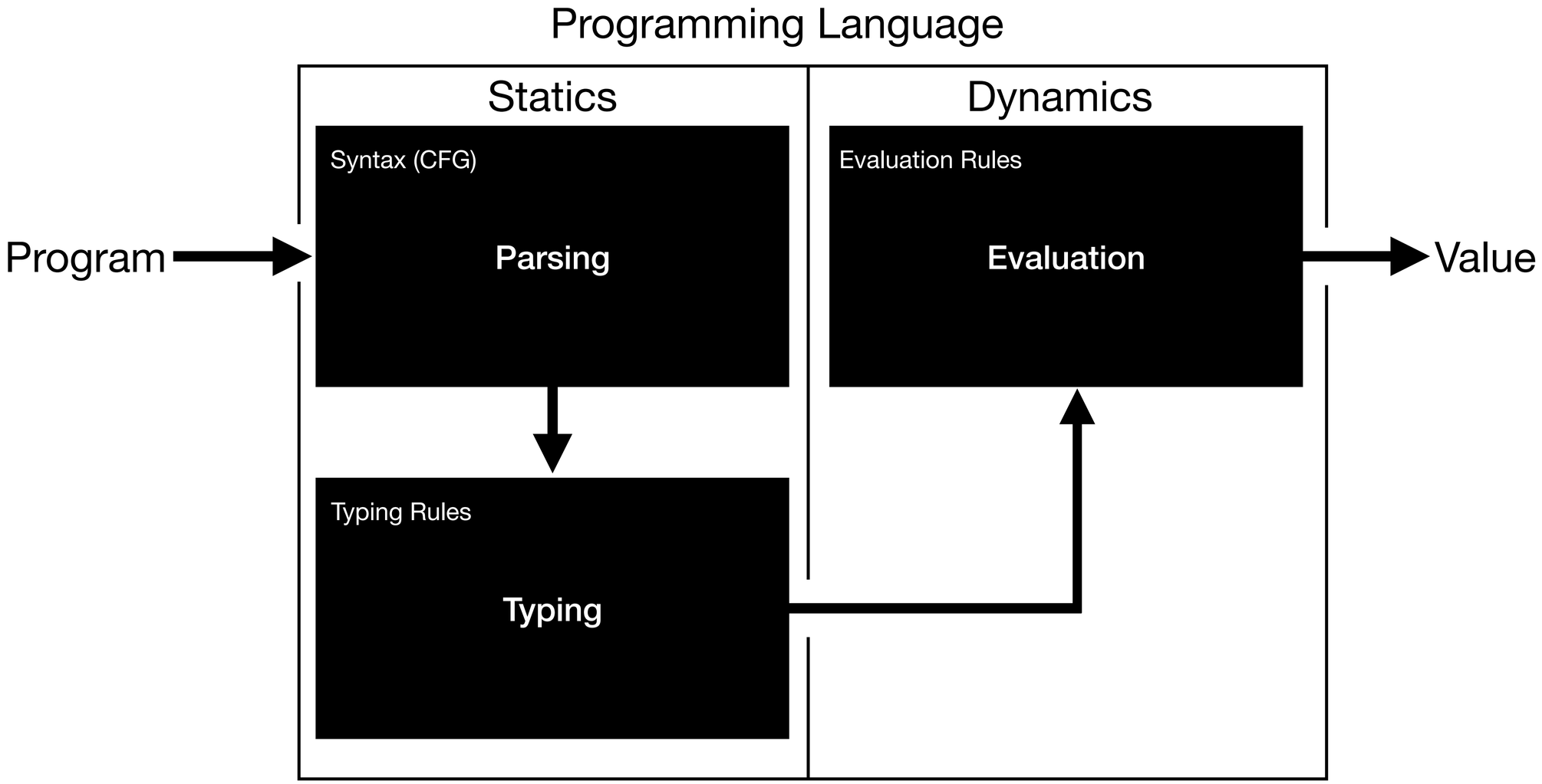
Object-Oriented Programming

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

$$
\begin{aligned}
p ::= \; & x \\
\mid \; & () \\
\mid \; & \mathsf{T} \\
\mid \; & \mathsf{F} \\
\mid \; & 0 \\
\mid \; & \mathsf{succ}(x)
\end{aligned}
$$

Nats

$$
\begin{aligned}
n ::= \; & 0 \\
\mid \; & \mathsf{succ}(n)
\end{aligned}
$$

Terms

$$
\begin{aligned}
t ::= \; & \mathsf{T} \\
\mid \; & \mathsf{F} \\
\mid \; & 0 \\
\mid \; & \mathsf{succ}(t) \\
\mid \; & x \\
\mid \; & \mathsf{fun}\, x : T \to \{t\} \\
\mid \; & t_1\, t_2 \\
\mid \; & () \\
\mid \; & \mathsf{match}\, t\, \{ \mid p_1 \to t_1 \mid \ldots \mid p_i \to t_i \}
\end{aligned}
$$

Values

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

Types

$$
\begin{aligned}
T ::= \; & \mathsf{Bool} \\
\mid \; & \mathsf{Nat} \\
\mid \; & () \\
\mid \; & T_1 \to T_2
\end{aligned}
$$

Statics: Bools

$$\frac{\Gamma \vdash t_1 : \mathsf{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathsf{match}\, t\, \{ \mid \mathsf{T} \to t_2 \mid \mathsf{F} \to t_3 \} : T}\ \mathsf{If}$$

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

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

Statics: Nats

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

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

Statics: Functions

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

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

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

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \mathsf{match}\, t_1\, \{\ |\ x \to t_2\} : T}\ \text{Let}$$

Statics: Unit

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

Call-by-Value Dynamics: Match

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

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

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

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

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

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

Call-by-Value Dynamics: Match

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

Call-by-Value Dynamics: Functions

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

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

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

Part 2:Unit

Example: Side Effects

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

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

Statics: Unit

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

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

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

$$t_1; t_2 = \text{match}\ t_1\ \{\ |\ () \to t_2\}$$

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

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

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

Sequencing is match!

Call-by-Value Dynamics: Unit

$$\frac{}{\mathsf{match}\,()\{\ |\ ()\to t_2\}\rightsquigarrow t_2}\ \mathrm{Unit}\beta$$

$$\frac{t_1\rightsquigarrow t_1'}{\mathsf{match}\,t_1\,\{\ |\ ()\to t_2\}\rightsquigarrow \mathsf{match}\,t_1'\,\{\ |\ ()\to t_2\}}\ \mathrm{Match}$$

Part 3:Pairs

New Syntactic Forms: Adding Pairs

Terms

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

Values

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

Types

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

Example Programs

$$\begin{array}{l}\text{let twist} = \text{fun}\, p : (\text{Bool}, \text{Bool}) \to \{(p.2, p.1)\} \\ \qquad\qquad \text{in twist}\,(\text{T}, \text{F})\end{array}$$

$$\begin{array}{l}\text{let second} = \text{fun}\, p : (\text{Bool} \to \text{Bool}, \text{Bool}) \to \{\text{if}\, p.2\, \text{then}\, p.1\, \text{T}\, \text{else F}\} \\ \qquad\qquad\qquad \text{in second}\,(\text{fun}\, x : \text{Bool} \to \{\text{if}\, x\, \text{then F else T}\}, \text{T})\end{array}$$

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

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

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

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

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

Statics: Pairs

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

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

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

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

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

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

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

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

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

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

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

Values

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

Types

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

Example: Tuple

(T, F, T, F, F)

```
fun (p : (Bool, Bool, Bool)) → {
    match p {
        (x, y, z) → if x
                    then if y
                         then z
                         else False
                    else False
    }
}
```

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

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

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

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

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

Statics: Tuples

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

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

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

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

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

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

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

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

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

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

Values

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

Types

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

Example: Records

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{t \rightsquigarrow t'}{t \,.\, l_i \rightsquigarrow t' \,.\, l_i}\ \text{Proj} \qquad \frac{}{(l_1 = v_1, \ldots, l_i = v_i) \,.\, l_j \rightsquigarrow v_j}\ \text{Record}\beta$$

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

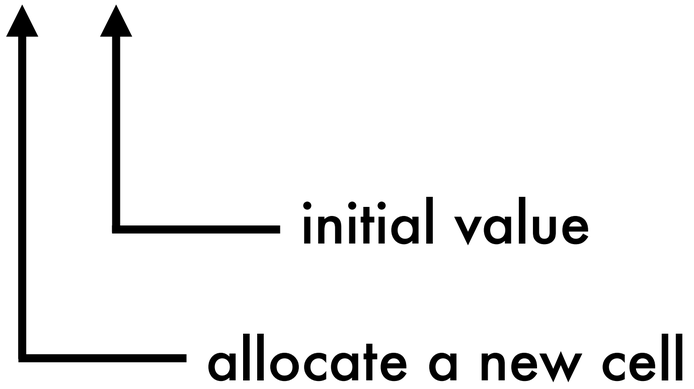
Allocating a reference: ref 5 : Ref Nat

Allocation

Allocating a reference: ref 5 : Ref Nat

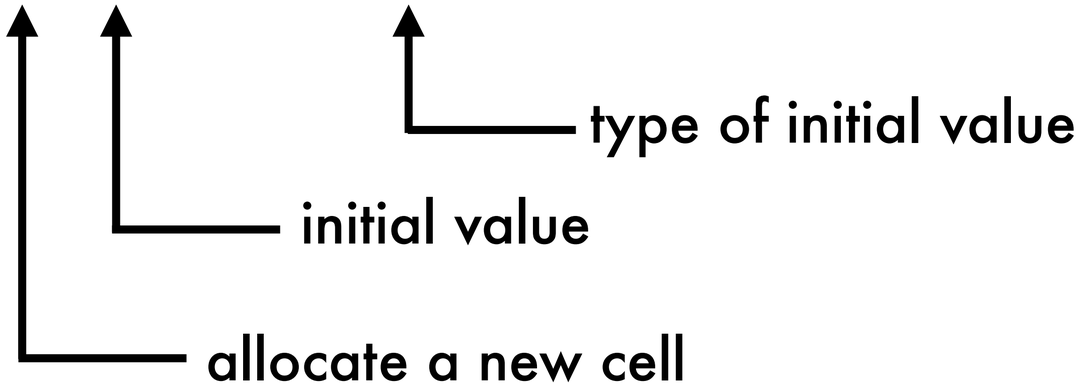
Allocation

Allocating a reference: ref 5 : Ref Nat



Allocation

Allocating a reference: ref 5 : Ref Nat

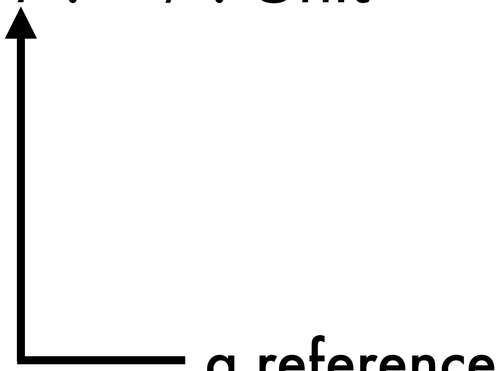


Assignment

Assignment operator: $r := 7$: Unit

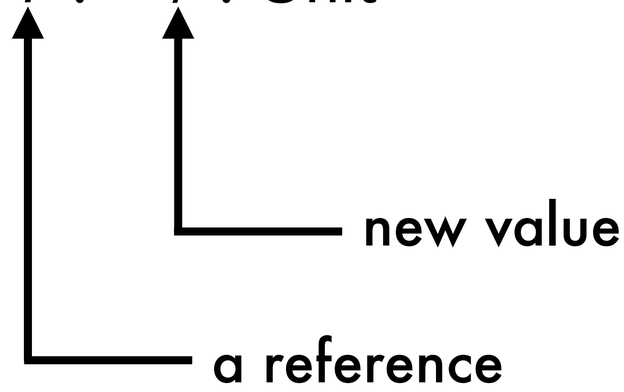
Assignment

Assignment operator: $r := 7$: Unit



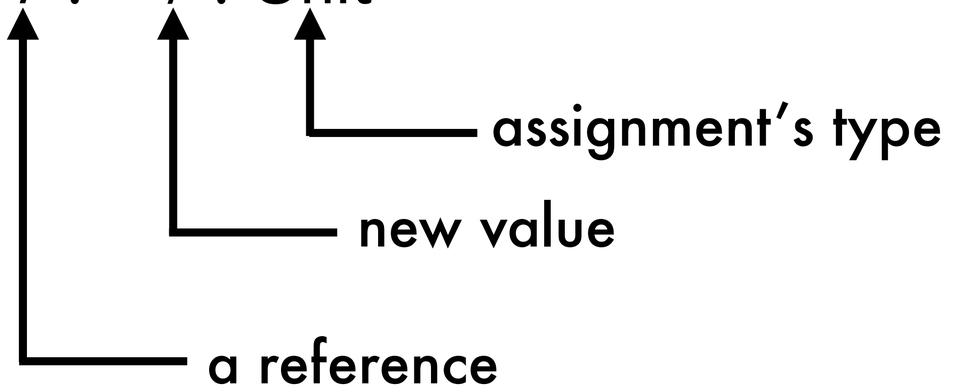
Assignment

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



Assignment

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



Assignment

Assignment operator: $r := 7$: Unit

Example:

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

$\#\, r : \text{Ref Nat}$

$r := 7$

$\#\, \text{unit} : \text{Unit}$

Dereferencing

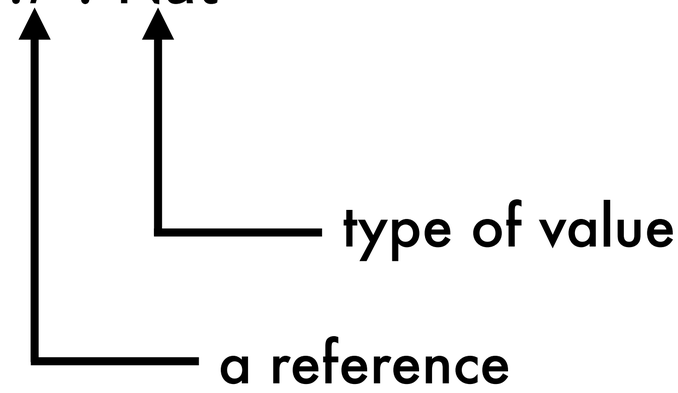
Dereferencing operator: $!r$: Nat

Dereferencing

Dereferencing operator: $!r$: Nat

Dereferencing

Dereferencing operator: $!r$: Nat



Dereferencing

Dereferencing operator: $!r$: Nat

Example:

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

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

$!r$

$\# 5 : \text{Nat}$

$r := 7$

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

$!r$

$\# 7 : \text{Nat}$

Sequencing

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

Sequencing

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

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

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

$\#8 : \text{Nat}$

Sequencing

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

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

Stores

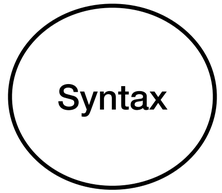
Part 7: Subtyping

Part 8: Imperative Objects

Part 9: OOP in OCaml

What is OCaml?

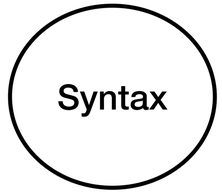
Core Design Concepts:



An object oriented, imperative, functional programming language.

What is OCaml?

Core Design Concepts:

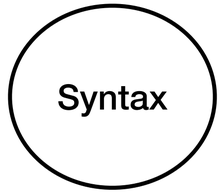


An object oriented, imperative, functional programming language.

OCaml mixes all of these paradigms together.

What is OCaml?

Core Design Concepts:

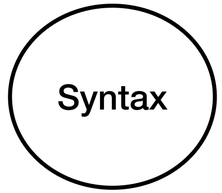


An object oriented, imperative, functional programming language.

OCaml mixes all of these paradigms together.

What is OCaml?

Core Design Concepts:

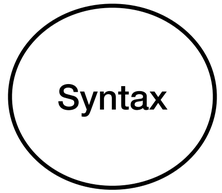


An object oriented, imperative, functional programming language.

OCaml mixes all of these paradigms together.

What is OCaml?

Core Design Concepts:

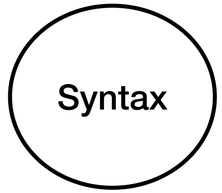


An object oriented, imperative, functional programming language.

OCaml mixes all of these paradigms together.

What is OCaml?

Core Design Concepts:

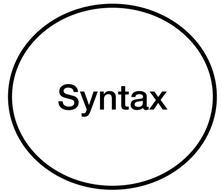


An object oriented, imperative, functional programming language.

OCaml mixes all of these paradigms together.

Class Definitions

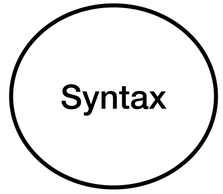
Core Design Concepts:




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

Class Definitions

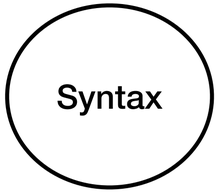
Core Design Concepts:




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

Class Definitions

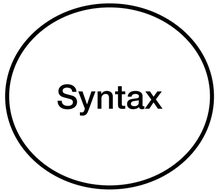
Core Design Concepts:




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

Class Definitions

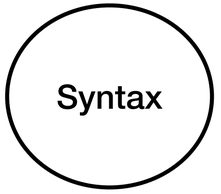
Core Design Concepts:




```
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

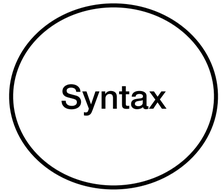
Core Design Concepts:




```
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

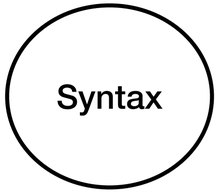
Core Design Concepts:




```
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

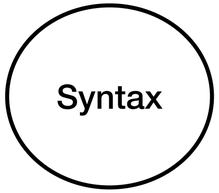
Core Design Concepts:




```
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

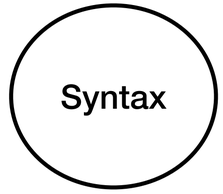
Core Design Concepts:




```
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:

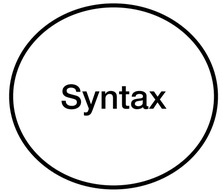



```
# let s = new stack_of_ints;;

val s : stack_of_ints = <obj>
```

Accessing fields and methods

Core Design Concepts:

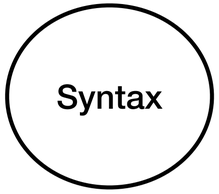



```
s#fieldName

s#methodName
```

Accessing fields and methods

Core Design Concepts:




```
# 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 = ()
```