

CMSC 330: Organization of Programming Languages

Operational Semantics

Formal Semantics of a Prog. Lang.

- ▶ Mathematical description of the meaning of programs written in that language
 - What a program computes, and what it does

"1+2"

Concrete
Syntax



Parse

Plus (Int 1, Int 2)

Abstract
Syntax

- ▶ What does **Plus (Int 1, Int 2)** mean?

1+2

Operational semantics

- ▶ Define how programs execute
 - Often on an abstract machine (mathematical model of computer)
 - Analogous to interpretation
- ▶ We will define an operational semantics for Micro-Ocaml
 - And develop an interpreter for it, along the way
- ▶ Approach: use **rules** to define a **judgment**

$$e \Rightarrow v$$

Micro-OCaml Expression Grammar

$$e ::= x \mid n \mid e + e \mid \text{let } x = e1 \text{ in } e2$$

Corresponding AST:

```
type id = string
type exp =
  | Ident of id           (* x *)
  | Num of int            (* n *)
  | Plus of exp * exp     (* e+e *)
  | Let of id * exp * exp (* let x=e1 in e2 *)
```

Defining the Semantics

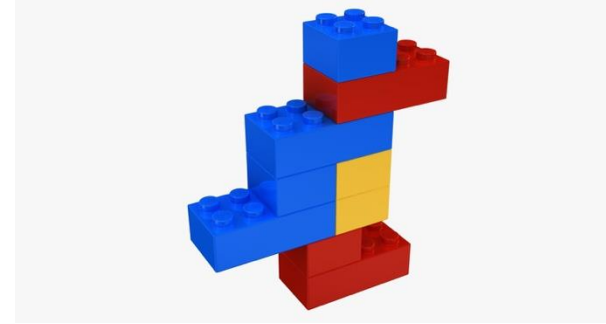
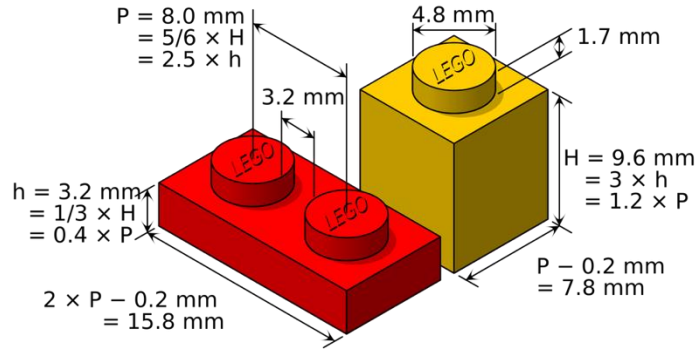
- ▶ Use **rules** to define **judgment** $e \Rightarrow v$
- ▶ Inference Rules

$$\frac{H_1, H_2 \dots H_n}{C}$$

$$\frac{\forall x (Man(x) \rightarrow Mortal(x)) \quad Man(Socrates)}{\therefore Mortal(Socrates)}$$

$$H_1 \wedge H_2 \wedge \dots H_n \Rightarrow C$$

Rules are Lego Blocks



Rules of Inference: Num and Sum

$n \Rightarrow n$

axiom

$e1 \Rightarrow n1 \quad e2 \Rightarrow n2 \quad n3 \text{ is } n1+n2$

$e1 + e2 \Rightarrow n3$

```
match e with
| Num n -> n
| Plus (e1,e2) ->
    let n1 = eval e1 in
    let n2 = eval e2 in
    let n3 = n1 + n2 in
    n3
```

Rules of Inference: Let

$$e1 \Rightarrow v1 \quad e2\{v1/x\} \Rightarrow v2$$
$$\text{let } x = e1 \text{ in } e2 \Rightarrow v2$$

```
match e with
| Let (x,e1,e2) ->
    let v1 = eval e1 in
    let e2' = subst v1 x e2 in
    let v2 = eval e2'
    in v2
```


Derivations

- ▶ When we apply rules to an expression in succession, we produce a **derivation**
 - It's a kind of **tree**, rooted at the conclusion
- ▶ Produce a derivation by **goal-directed search**
 - Pick a rule that could prove the goal
 - Then repeatedly apply rules on the corresponding hypotheses

➤ **Goal:** Show that `let x = 4 in x+3` \Rightarrow 7

$$\begin{array}{r} 4 \Rightarrow 4 \quad 3 \Rightarrow 3 \quad 7 \Rightarrow 4+3 \\ \hline 4 \Rightarrow 4 \quad 4+3 \Rightarrow 7 \\ \hline \text{let } x = 4 \text{ in } x+3 \Rightarrow 7 \end{array}$$

Derivations

	$e1 \Rightarrow n1 \quad e2 \Rightarrow n2 \quad n3 \text{ is } n1+n2$
$n \Rightarrow n$	$e1 + e2 \Rightarrow n3$

$e1 \Rightarrow v1 \quad e2\{v1/x\} \Rightarrow v2$
$\text{let } x = e1 \text{ in } e2 \Rightarrow v2$

Goal: show that

$\text{let } x = 4 \text{ in } x+3 \Rightarrow 7$

	$4 \Rightarrow 4 \quad 3 \Rightarrow 3 \quad 7 \text{ is } 4+3$
$4 \Rightarrow 4$	$4+3 \Rightarrow 7$
$\text{let } x = 4 \text{ in } x+3 \Rightarrow 7$	

Quiz 1

What is derivation of the following judgment?

$$2 + (3 + 8) \Rightarrow 13$$

~~(a)~~

$$\begin{array}{rcl} 2 \Rightarrow 2 & 3 + 8 \Rightarrow 11 & \\ \hline 2 + (3 + 8) \Rightarrow 13 \end{array}$$

(b)

$$\begin{array}{rcl} & 8 \Rightarrow 8 & \\ & 3 \Rightarrow 3 & \\ & 11 \text{ is } 3+8 & \\ & \hline 2 \Rightarrow 2 & 3 + 8 \Rightarrow 11 & 13 \text{ is } 2+11 \\ \hline 2 + (3 + 8) \Rightarrow 13 \end{array}$$

out of order

(c)

$$\begin{array}{rcl} 3 \Rightarrow 3 & 8 \Rightarrow 8 & \\ \hline 3 + 8 \Rightarrow 11 & & 2 \Rightarrow 2 \\ \hline 2 + (3 + 8) \Rightarrow 13 \end{array}$$

Quiz 1

What is derivation of the following judgment?

$$2 + (3 + 8) \Rightarrow 13$$

(a)

$$\begin{array}{rcl} 2 \Rightarrow 2 & 3 + 8 \Rightarrow 11 & \\ \hline 2 + (3 + 8) \Rightarrow 13 \end{array}$$

(b)

$$\begin{array}{rcl} & 8 \Rightarrow 8 & \\ & 3 \Rightarrow 3 & \\ & 11 \text{ is } 3+8 & \\ & \hline 2 \Rightarrow 2 & 3 + 8 \Rightarrow 11 & 13 \text{ is } 2+11 \\ \hline 2 + (3 + 8) \Rightarrow 13 \end{array}$$

(c)

$$\begin{array}{rcl} 3 \Rightarrow 3 & 8 \Rightarrow 8 & \\ \hline 3 + 8 \Rightarrow 11 & & 2 \Rightarrow 2 \\ \hline 2 + (3 + 8) \Rightarrow 13 \end{array}$$

Definitional Interpreter

- ▶ The style of rules lends itself directly to the implementation of an **interpreter as a recursive function**

```
let rec eval (e:exp):value =  
  match e with  
    Ident x -> (* no rule *)  
      failwith "no value"  
  | Num n -> n  
  | Plus (e1,e2) ->  
    let n1 = eval e1 in  
    let n2 = eval e2 in  
    let n3 = n1+n2 in  
    n3  
  | Let (x,e1,e2) ->  
    let v1 = eval e1 in  
    let e2' = subst v1 x e2 in  
    let v2 = eval e2' in v2
```

$$n \Rightarrow n$$

$$\frac{e1 \Rightarrow n1 \quad e2 \Rightarrow n2 \quad n3 \text{ is } n1+n2}{e1 + e2 \Rightarrow n3}$$

$$\frac{e1 \Rightarrow v1 \quad e2\{v1/x\} \Rightarrow v2}{\text{let } x = e1 \text{ in } e2 \Rightarrow v2}$$

Derivations = Interpreter Call Trees

$$\frac{\frac{4 \Rightarrow 4}{\text{let } x = 4 \text{ in } x+3 \Rightarrow 7} \quad \frac{3 \Rightarrow 3 \quad 7 \text{ is } 4+3}{4+3 \Rightarrow 7}}{}$$

Has the same shape as the recursive call tree of the interpreter:

$$\frac{\frac{\text{eval Num } 4 \Rightarrow 4 \quad \text{eval Num } 3 \Rightarrow 3 \quad 7 \text{ is } 4+3}{\text{eval (subst 4 "x" Plus (Ident ("x"), Num 3))} \Rightarrow 7} \quad \text{eval Num } 4 \Rightarrow 4}{\text{eval Let ("x", Num 4, Plus (Ident ("x"), Num 3))} \Rightarrow 7}$$

Semantics Defines Program Meaning

- ▶ $e \Rightarrow v$ holds if and only if a *proof* can be built
 - Proofs are derivations: axioms at the top, then rules whose hypotheses have been proved to the bottom
 - No proof means there *exists no* v for which $e \Rightarrow v$
- ▶ Proofs can be constructed bottom-up
 - In a goal-directed fashion
- ▶ Thus, function $\text{eval } e = \{v \mid e \Rightarrow v\}$
 - Determinism of semantics implies at most one element for any e
- ▶ So: Expression e *means* v

Environment-style Semantics

- ▶ So far, semantics used substitution to handle variables
 - As we evaluate, we replace all occurrences of a variable **x** with values it is bound to
- ▶ An alternative semantics, closer to a real implementation, is to use an **environment**
 - As we evaluate, we maintain an explicit map from variables to values, and look up variables as we see them

Environments

- ▶ Mathematically, an environment is a **partial function** from identifiers to values
 - If A is an environment, and x is an identifier, then $A(x)$ can either be
 - a value v (intuition: the value of the variable stored on the stack)
 - undefined (intuition: the variable has not been declared)
- ▶ An environment can be visualized as a table
 - If A is

Id	Val
x	0
y	2

- then $A(x)$ is 0, $A(y)$ is 2, and $A(z)$ is undefined

Notation, Operations on Environments

- ▶ \bullet is the empty environment
- ▶ $A, \mathbf{x}:\mathbf{v}$ is the environment that extends A with a mapping from \mathbf{x} to \mathbf{v}
 - Sometimes just write $\mathbf{x}:\mathbf{v}$ instead of $\bullet, \mathbf{x}:\mathbf{v}$ for brevity
- ▶ Lookup $A(\mathbf{x})$ is defined as follows
$$\begin{aligned} \bullet(\mathbf{x}) &= \text{undefined} \\ (A, \mathbf{y}:\mathbf{v})(\mathbf{x}) &= \begin{cases} \mathbf{v} & \text{if } \mathbf{x} = \mathbf{y} \\ A(\mathbf{x}) & \text{if } \mathbf{x} \neq \mathbf{y} \text{ and } A(\mathbf{x}) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases} \end{aligned}$$

Definitional Interpreter: Environments

```
type env = (id * value) list

let extend env x v = (x,v)::env

let rec lookup env x =
  match env with
  | [] -> failwith "undefined"
  | (y,v)::env' ->
    if x = y then v
    else lookup env' x
```

An environment is just a list of mappings,
which are just pairs of variable to value
- called an **association list**

Semantics with Environments

- ▶ The environment semantics changes the judgment

$$e \Rightarrow v$$

to be

$$A; e \Rightarrow v$$

where A is an environment

- Idea: A is used to give values to the identifiers in e

Environment-style Rules

$$\frac{A(\mathbf{x}) = \mathbf{v}}{A; \mathbf{x} \Rightarrow \mathbf{v}}$$

Look up variable \mathbf{x} in environment A

$$\frac{}{A; \mathbf{n} \Rightarrow \mathbf{n}}$$

$$\frac{A; \mathbf{e1} \Rightarrow \mathbf{v1} \quad A, \mathbf{x}:\mathbf{v1}; \mathbf{e2} \Rightarrow \mathbf{v2}}{A; \text{let } \mathbf{x} = \mathbf{e1} \text{ in } \mathbf{e2} \Rightarrow \mathbf{v2}}$$

Extend environment A with mapping from \mathbf{x} to $\mathbf{v1}$

$$\frac{A; \mathbf{e1} \Rightarrow \mathbf{n1} \quad A; \mathbf{e2} \Rightarrow \mathbf{n2} \quad \mathbf{n3} \text{ is } \mathbf{n1} + \mathbf{n2}}{A; \mathbf{e1} + \mathbf{e2} \Rightarrow \mathbf{n3}}$$

Definitional Interpreter: Evaluation

```
let rec eval env e =  
  match e with  
    Ident x -> lookup env x  
  | Num n -> n  
  | Plus (e1,e2) ->  
    let n1 = eval env e1 in  
    let n2 = eval env e2 in  
    let n3 = n1+n2 in  
    n3  
  | Let (x,e1,e2) ->  
    let v1 = eval env e1 in  
    let env' = extend env x v1 in  
    let v2 = eval env' e2 in v2
```

Quiz 2

What is a derivation of the following judgment?

•; let $x=3$ in $x+2 \Rightarrow 5$

(a)

$$\frac{\begin{array}{c} x \Rightarrow 3 \quad 2 \Rightarrow 2 \quad 5 \text{ is } 3+2 \\ \hline 3 \Rightarrow 3 \quad x+2 \Rightarrow 5 \\ \hline \end{array}}{\text{let } x=3 \text{ in } x+2 \Rightarrow 5}$$

(c)

$$\frac{\begin{array}{c} x:2; x \Rightarrow 3 \quad x:2; 2 \Rightarrow 2 \quad 5 \text{ is } 3+2 \\ \hline \end{array}}{\bullet; \text{let } x=3 \text{ in } x+2 \Rightarrow 5}$$

(b)

$$\frac{\begin{array}{c} x:3; x \Rightarrow 3 \quad x:3; 2 \Rightarrow 2 \quad 5 \text{ is } 3+2 \\ \hline \bullet; 3 \Rightarrow 3 \quad x:3; x+2 \Rightarrow 5 \\ \hline \end{array}}{\bullet; \text{let } x=3 \text{ in } x+2 \Rightarrow 5}$$

Quiz 2

What is a derivation of the following judgment?

•; let $x=3$ in $x+2 \Rightarrow 5$

(a)

$$\frac{\frac{x \Rightarrow 3 \quad 2 \Rightarrow 2 \quad 5 \text{ is } 3+2}{\text{-----}} \quad \frac{3 \Rightarrow 3 \quad x+2 \Rightarrow 5}{\text{-----}}}{\text{let } x=3 \text{ in } x+2 \Rightarrow 5}$$

(c)

$$\frac{x:2; x \Rightarrow 3 \quad x:2; 2 \Rightarrow 2 \quad 5 \text{ is } 3+2 \quad \text{----}}{\text{-----}} \quad \bullet; \text{let } x=3 \text{ in } x+2 \Rightarrow 5$$

(b)

$$\frac{\bullet; 3 \Rightarrow 3 \quad \frac{x:3; x \Rightarrow 3 \quad x:3; 2 \Rightarrow 2 \quad 5 \text{ is } 3+2}{\text{-----}} \quad x:3; x+2 \Rightarrow 5}{\text{-----}} \quad \bullet; \text{let } x=3 \text{ in } x+2 \Rightarrow 5$$

Adding Conditionals to Micro-OCaml

```

$$e ::= x \mid v \mid e + e \mid \text{let } x = e \text{ in } e$$

$$\mid \text{eq0 } e \mid \text{if } e \text{ then } e \text{ else } e$$

$$v ::= n \mid \text{true} \mid \text{false}$$

```

- In terms of interpreter definitions:

```
type exp =  
  | Val of value  
  | ... (* as before *)  
  | Eq0 of exp  
  | If of exp * exp * exp  
  
type value =  
  | Int of int  
  | Bool of bool
```

Rules for Eq0 and Booleans

$A; \text{true} \Rightarrow \text{true}$

$A; \text{false} \Rightarrow \text{false}$

$A; e \Rightarrow 0$

$A; \text{eq0 } e \Rightarrow \text{true}$

$A; e \Rightarrow v \quad v \neq 0$

$A; \text{eq0 } e \Rightarrow \text{false}$

Rules for Conditionals

$$A; e1 \Rightarrow \text{true} \quad A; e2 \Rightarrow v$$
$$A; \text{if } e1 \text{ then } e2 \text{ else } e3 \Rightarrow v$$
$$A; e1 \Rightarrow \text{false} \quad A; e3 \Rightarrow v$$
$$A; \text{if } e1 \text{ then } e2 \text{ else } e3 \Rightarrow v$$

- Notice that only one branch is evaluated

Quiz 3

What is the derivation of the following judgment?

•; if eq0 3-2 then 5 else 10 \Rightarrow 10

(a)

```
•; 3  $\Rightarrow$  3    •; 2  $\Rightarrow$  2    3-2 is 1
-----
•; eq0 3-2  $\Rightarrow$  false      •; 10  $\Rightarrow$  10
-----
•; if eq0 3-2 then 5 else 10  $\Rightarrow$  10
```

(b)

```
3  $\Rightarrow$  3    2  $\Rightarrow$  2
3-2 is 1
-----
eq0 3-2  $\Rightarrow$  false      10  $\Rightarrow$  10
-----
if eq0 3-2 then 5 else 10  $\Rightarrow$  10
```

(c)

```
•; 3  $\Rightarrow$  3
•; 2  $\Rightarrow$  2
3-2 is 1
-----
•; 3-2  $\Rightarrow$  1      1  $\neq$  0
-----
•; eq0 3-2  $\Rightarrow$  false      •; 10  $\Rightarrow$  10
-----
•; if eq0 3-2 then 5 else 10  $\Rightarrow$  10
```

Quiz 3

What is the derivation of the following judgment?

•; if eq0 3-2 then 5 else 10 \Rightarrow 10

(a)

```
•; 3  $\Rightarrow$  3    •; 2  $\Rightarrow$  2    3-2 is 1
-----
•; eq0 3-2  $\Rightarrow$  false          •; 10  $\Rightarrow$  10
-----
•; if eq0 3-2 then 5 else 10  $\Rightarrow$  10
```

(b)

```
3  $\Rightarrow$  3    2  $\Rightarrow$  2
3-2 is 1
-----
eq0 3-2  $\Rightarrow$  false          10  $\Rightarrow$  10
-----
if eq0 3-2 then 5 else 10  $\Rightarrow$  10
```

(c)

```
•; 3  $\Rightarrow$  3
•; 2  $\Rightarrow$  2
3-2 is 1
-----
•; 3-2  $\Rightarrow$  1    1  $\neq$  0
-----
•; eq0 3-2  $\Rightarrow$  false          •; 10  $\Rightarrow$  10
-----
•; if eq0 3-2 then 5 else 10  $\Rightarrow$  10
```

Updating the Interpreter

```
let rec eval env e =  
  match e with  
    Ident x -> lookup env x  
  | Val v -> v  
  | Plus (e1,e2) ->  
    let Int n1 = eval env e1 in  
    let Int n2 = eval env e2 in  
    let n3 = n1+n2 in  
    Int n3  
  | Let (x,e1,e2) ->  
    let v1 = eval env e1 in  
    let env' = extend env x v1 in  
    let v2 = eval env' e2 in v2  
  | Eq0 e1 ->  
    let Int n = eval env e1 in  
    if n=0 then Bool true else Bool false  
  | If (e1,e2,e3) ->  
    let Bool b = eval env e1 in  
    if b then eval env e2  
    else eval env e3
```

Adding Closures to Micro-OCaml

$e ::= x \mid v \mid e + e \mid \text{let } x = e \text{ in } e$
 $\mid \text{eq0 } e \mid \text{if } e \text{ then } e \text{ else } e$
 $\mid ee \mid \text{fun } x \rightarrow e$

$v ::= n \mid \text{true} \mid \text{false} \mid (A, \lambda x. e)$

Environment

Code
(id and exp)

- In terms of interpreter definitions:

```
type exp =  
  | Val of value  
  | If of exp * exp * exp  
  ... (* as before *)  
  | Call of exp * exp  
  | Fun of id * exp
```

```
type value =  
  Int of int  
  | Bool of bool  
  | Closure of env * id * exp
```

Rule for Closures: Lexical/Static Scoping

$$A; \text{fun } x \rightarrow e \Rightarrow (A, \lambda x. e)$$

$$\frac{A; e1 \Rightarrow (A', \lambda x. e) \quad A; e2 \Rightarrow v1 \quad A', x:v1; e \Rightarrow v}{A; e1 \ e2 \Rightarrow v}$$

► Notice

- Creating a closure captures the current environment A
- A call to a function
 - evaluates the body of the closure's code e with function closure's environment A' extended with parameter x bound to argument $v1$

Rule for Closures: Dynamic Scoping

$$A; \text{fun } x \rightarrow e \Rightarrow (\bullet, \lambda x. e)$$
$$\frac{A; e1 \Rightarrow (\bullet, \lambda x. e) \quad A; e2 \Rightarrow v1 \quad A, x:v1; e \Rightarrow v}{A; e1 \ e2 \Rightarrow v}$$

► Notice

- Creating a closure ignores the current environment A
- A call to a function
 - evaluates the body of the closure's code e with the current environment A extended with parameter x bound to argument $v1$

Scaling up

- ▶ Operational semantics can handle full languages
 - With records, recursive variant types, objects, first-class functions, and more
- ▶ Provides a concise notation for explaining what a language does. Clearly shows:
 - Evaluation order
 - Call-by-value vs. call-by-name
 - Static scoping vs. dynamic scoping
 - ... We may look at more of these later

Scaling up: Lego City

